

Your title here

Generated by Doxygen 1.9.1

0.1 SMT-RAT	1
0.2 Installation	1
0.2.1 Requirements	1
0.2.2 Download	2
0.2.3 Configuration	2
0.2.4 Build	2
0.2.5 Check build	2
0.3 System architecture	3
0.3.1 Different libraries	3
0.3.2 Software design	3
0.3.3 Modules	4
0.3.4 Strategy	4
0.3.5 Manager	4
0.3.6 Procedures implemented as modules	4
0.3.7 Infeasible subsets and lemmas	5
0.4 Modules	5
0.4.1 Main members of a module	5
0.4.2 Interfaces to implement	5
0.4.2.1 Informing about a constraint	6
0.4.2.2 Adding a received formula	6
0.4.2.3 Removing a received formula	6
0.4.2.4 Checking for satisfiability	6
0.4.2.5 Updating the model/satisfying assignment	7
0.4.3 Running backend modules	7
0.4.4 Auxiliary functions	8
0.4.5 Existing module implementation	9
0.4.6 Available modules	9
0.4.6.1 BEModule	9
0.4.6.2 BVModule	9
0.4.6.3 CNFerModule	9
0.4.6.4 CSplitModule	9
0.4.6.5 CoCoAGBModule	10
0.4.6.6 CubeLIAModule	10
0.4.6.7 CurryModule	10
0.4.6.8 EMModule	10
0.4.6.9 EModule	10
0.4.6.10 FPPModule	10
0.4.6.11 FouMoModule	10
0.4.6.12 GBModule	11
0.4.6.13 GBPPModule	11
0.4.6.14 ICEModule	11
0.4.6.15 ICPModule	11

0.4.6.16 IncWidthModule	12
0.4.6.17 IntBlastModule	12
0.4.6.18 IntEqModule	12
0.4.6.19 LRAModule	12
0.4.6.20 LVEModule	12
0.4.6.21 MCBModule	13
0.4.6.22 NRAILModule	13
0.4.6.23 NewCADModule	13
0.4.6.24 PBGaussModule	13
0.4.6.25 PBPPModule	13
0.4.6.26 PFEModule	13
0.4.6.27 SATModule	14
0.4.6.28 STropModule	14
0.4.6.29 SplitSOSModule	14
0.4.6.30 SymmetryModule	14
0.4.6.31 UFCegarModule	15
0.4.6.32 UnionFindModule	15
0.4.6.33 VSModule	15
0.5 Strategies	15
0.6 Using SMT-RAT	16
0.6.1 Standalone solver	17
0.6.1.1 Formula analysis	18
0.6.1.2 Preprocessing	18
0.6.1.3 Quantifier elimination	18
0.6.1.4 DIMACS solving	18
0.6.1.5 Pseudo-Boolean solving	18
0.6.1.6 Optimization	18
0.6.2 Embedding in other software	18
0.6.2.1 Interface	19
0.6.2.2 Syntax of formulas	20
0.6.2.3 Boolean combinations of constraints and Boolean variables	20
0.6.2.4 Normalized constraints	22
0.6.2.5 Linking	22
0.7 Other tools	22
0.7.1 Benchmarking	22
0.7.2 Delta debugging	22
0.7.2.1 SMT-RAT's own delta tool	23
0.7.2.2 Using ddSMT	23
0.7.3 Analyzer	23
0.7.4 Preprocessing	23
0.7.5 Benchmax	23
0.7.5.1 General usage	24

0.7.5.2 Tools	24
0.7.5.3 Backends	25
0.7.5.4 Troubleshooting	25
0.7.5.5 Benchmax python utility	25
0.7.6 Delta	27
0.7.6.1 Delta debugging	27
0.8 Developers information	28
0.8.1 Code style	28
0.8.2 Documentation	28
0.8.2.1 Code comments	29
0.8.2.2 Writing out-of-source documentation	30
0.8.3 Settings	30
0.8.4 Logging	30
0.8.5 Statistics and timing	31
0.8.6 Testing	33
0.8.7 Validation	33
0.8.8 Checkpoints	34
0.8.9 Finding and Reporting Bugs	35
0.9 README	35
0.10 NewCoveringModule #and	36
0.10.1 Introduction	36
0.10.2 Usage	36
0.10.3 Efficiency	36
0.11 Todo List	36
0.12 Hierarchical Index	37
0.12.1 Class Hierarchy	37
0.13 Data Structure Index	50
0.13.1 Data Structures	50
0.14 Namespace Documentation	64
0.14.1 benchmax Namespace Reference	64
0.14.1.1 Typedef Documentation	66
0.14.1.2 Function Documentation	67
0.14.2 benchmax::settings Namespace Reference	70
0.14.2.1 Function Documentation	71
0.14.3 benchmax::slurm Namespace Reference	72
0.14.3.1 Function Documentation	73
0.14.4 benchmax::ssh Namespace Reference	75
0.14.5 carl Namespace Reference	75
0.14.6 delta Namespace Reference	75
0.14.6.1 Typedef Documentation	76
0.14.6.2 Function Documentation	76
0.14.7 Minisat Namespace Reference	79

0.14.7.1 Typedef Documentation	80
0.14.7.2 Function Documentation	81
0.14.7.3 Variable Documentation	84
0.14.8 smrat Namespace Reference	85
0.14.8.1 Detailed Description	92
0.14.8.2 Typedef Documentation	93
0.14.8.3 Enumeration Type Documentation	96
0.14.8.4 Function Documentation	98
0.14.8.5 Variable Documentation	106
0.14.9 smrat::analyzer Namespace Reference	106
0.14.9.1 Function Documentation	106
0.14.10 smrat::cad Namespace Reference	107
0.14.10.1 Typedef Documentation	110
0.14.10.2 Enumeration Type Documentation	110
0.14.10.3 Function Documentation	112
0.14.11 smrat::cad::debug Namespace Reference	114
0.14.12 smrat::cad::full Namespace Reference	114
0.14.12.1 Typedef Documentation	114
0.14.13 smrat::cad::full_ec Namespace Reference	114
0.14.13.1 Typedef Documentation	115
0.14.14 smrat::cad::preprocessor Namespace Reference	115
0.14.14.1 Function Documentation	115
0.14.15 smrat::cad::projection Namespace Reference	115
0.14.15.1 Function Documentation	116
0.14.16 smrat::cad::projection::brown Namespace Reference	117
0.14.16.1 Detailed Description	117
0.14.16.2 Function Documentation	118
0.14.17 smrat::cad::projection::collins Namespace Reference	118
0.14.17.1 Detailed Description	118
0.14.17.2 Function Documentation	118
0.14.18 smrat::cad::projection::hong Namespace Reference	119
0.14.18.1 Detailed Description	119
0.14.18.2 Function Documentation	119
0.14.19 smrat::cad::projection::lazard Namespace Reference	119
0.14.19.1 Detailed Description	120
0.14.19.2 Function Documentation	120
0.14.20 smrat::cad::projection::mccallum Namespace Reference	120
0.14.20.1 Detailed Description	120
0.14.20.2 Function Documentation	120
0.14.21 smrat::cad::projection::mccallum_partial Namespace Reference	121
0.14.21.1 Detailed Description	121
0.14.21.2 Function Documentation	121

0.14.22 smtrat::cad::projection_compare Namespace Reference	121
0.14.22.1 Typedef Documentation	122
0.14.22.2 Function Documentation	122
0.14.23 smtrat::cad::sample_compare Namespace Reference	123
0.14.23.1 Detailed Description	124
0.14.23.2 Typedef Documentation	124
0.14.23.3 Function Documentation	124
0.14.24 smtrat::cad::variable_ordering Namespace Reference	125
0.14.24.1 Function Documentation	125
0.14.25 smtrat::cadcells Namespace Reference	125
0.14.25.1 Detailed Description	126
0.14.25.2 Typedef Documentation	126
0.14.25.3 Variable Documentation	127
0.14.26 smtrat::cadcells::algorithms Namespace Reference	127
0.14.26.1 Detailed Description	128
0.14.26.2 Function Documentation	128
0.14.27 smtrat::cadcells::datastructures Namespace Reference	129
0.14.27.1 Detailed Description	132
0.14.27.2 Typedef Documentation	133
0.14.27.3 Function Documentation	133
0.14.28 smtrat::cadcells::datastructures::detail Namespace Reference	138
0.14.29 smtrat::cadcells::helper Namespace Reference	138
0.14.29.1 Function Documentation	139
0.14.30 smtrat::cadcells::operators Namespace Reference	139
0.14.30.1 Detailed Description	140
0.14.30.2 Enumeration Type Documentation	141
0.14.30.3 Function Documentation	141
0.14.30.4 Variable Documentation	143
0.14.31 smtrat::cadcells::operators::mccallum_filtered_impl Namespace Reference	143
0.14.31.1 Function Documentation	144
0.14.32 smtrat::cadcells::operators::properties Namespace Reference	144
0.14.32.1 Detailed Description	145
0.14.32.2 Function Documentation	145
0.14.33 smtrat::cadcells::operators::rules Namespace Reference	149
0.14.33.1 Detailed Description	150
0.14.33.2 Function Documentation	150
0.14.34 smtrat::cadcells::operators::rules::additional_root_outside_util Namespace Reference	154
0.14.34.1 Function Documentation	155
0.14.35 smtrat::cadcells::operators::rules::filter_util Namespace Reference	155
0.14.35.1 Enumeration Type Documentation	156
0.14.35.2 Function Documentation	156
0.14.36 smtrat::cadcells::operators::rules::ordering_util Namespace Reference	157

0.14.36.1 Typedef Documentation	157
0.14.36.2 Function Documentation	157
0.14.37 smrat::cadcells::representation Namespace Reference	157
0.14.37.1 Detailed Description	158
0.14.37.2 Typedef Documentation	158
0.14.37.3 Enumeration Type Documentation	158
0.14.37.4 Function Documentation	159
0.14.37.5 Variable Documentation	160
0.14.38 smrat::cadcells::representation::approximation Namespace Reference	160
0.14.38.1 Typedef Documentation	161
0.14.38.2 Enumeration Type Documentation	161
0.14.38.3 Function Documentation	162
0.14.39 smrat::cadcells::representation::util Namespace Reference	165
0.14.39.1 Function Documentation	165
0.14.40 smrat::compile_information Namespace Reference	167
0.14.40.1 Detailed Description	167
0.14.40.2 Variable Documentation	167
0.14.41 smrat::datastructures Namespace Reference	168
0.14.42 smrat::execution Namespace Reference	168
0.14.42.1 Enumeration Type Documentation	168
0.14.43 smrat::expression Namespace Reference	169
0.14.43.1 Typedef Documentation	169
0.14.43.2 Enumeration Type Documentation	169
0.14.43.3 Function Documentation	170
0.14.44 smrat::expression::simplifier Namespace Reference	171
0.14.44.1 Typedef Documentation	172
0.14.45 smrat::groebner Namespace Reference	172
0.14.45.1 Typedef Documentation	172
0.14.45.2 Function Documentation	172
0.14.46 smrat::helper Namespace Reference	172
0.14.46.1 Function Documentation	172
0.14.47 smrat::icp Namespace Reference	173
0.14.47.1 Typedef Documentation	173
0.14.47.2 Enumeration Type Documentation	173
0.14.47.3 Function Documentation	174
0.14.48 smrat::impl Namespace Reference	174
0.14.48.1 Function Documentation	175
0.14.48.2 Variable Documentation	175
0.14.49 smrat::ira Namespace Reference	175
0.14.49.1 Typedef Documentation	176
0.14.49.2 Function Documentation	176
0.14.49.3 Variable Documentation	181

0.14.50 smrat::lve Namespace Reference	181
0.14.50.1 Function Documentation	181
0.14.51 smrat::maxsmt Namespace Reference	182
0.14.51.1 Detailed Description	182
0.14.52 smrat::mcsat Namespace Reference	182
0.14.52.1 Typedef Documentation	183
0.14.52.2 Enumeration Type Documentation	184
0.14.52.3 Function Documentation	184
0.14.53 smrat::mcsat::arithmetic Namespace Reference	185
0.14.53.1 Function Documentation	185
0.14.54 smrat::mcsat::constraint_type Namespace Reference	186
0.14.54.1 Function Documentation	186
0.14.55 smrat::mcsat::fm Namespace Reference	187
0.14.55.1 Detailed Description	187
0.14.55.2 Function Documentation	188
0.14.56 smrat::mcsat::icp Namespace Reference	188
0.14.57 smrat::mcsat::nlsat Namespace Reference	188
0.14.58 smrat::mcsat::nlsat::helper Namespace Reference	188
0.14.58.1 Function Documentation	189
0.14.59 smrat::mcsat::onecell Namespace Reference	189
0.14.59.1 Typedef Documentation	190
0.14.59.2 Function Documentation	190
0.14.59.3 Variable Documentation	190
0.14.60 smrat::mcsat::onecellcad Namespace Reference	190
0.14.60.1 Typedef Documentation	192
0.14.60.2 Enumeration Type Documentation	192
0.14.60.3 Function Documentation	193
0.14.61 smrat::mcsat::onecellcad::levelwise Namespace Reference	197
0.14.62 smrat::mcsat::onecellcad::recursive Namespace Reference	197
0.14.62.1 Detailed Description	198
0.14.62.2 Enumeration Type Documentation	198
0.14.62.3 Function Documentation	199
0.14.62.4 Variable Documentation	199
0.14.63 smrat::mcsat::smtaf Namespace Reference	199
0.14.63.1 Typedef Documentation	199
0.14.63.2 Function Documentation	199
0.14.64 smrat::mcsat::variableordering Namespace Reference	199
0.14.64.1 Function Documentation	200
0.14.65 smrat::mcsat::variableordering::detail Namespace Reference	201
0.14.65.1 Function Documentation	201
0.14.66 smrat::mcsat::vs Namespace Reference	202
0.14.67 smrat::mcsat::vs::helper Namespace Reference	202

0.14.67.1 Function Documentation	203
0.14.68 smrat::onecellcad Namespace Reference	204
0.14.69 smrat::onecellcad::recursive Namespace Reference	204
0.14.69.1 Typedef Documentation	204
0.14.69.2 Function Documentation	205
0.14.70 smrat::options_detail Namespace Reference	206
0.14.70.1 Function Documentation	206
0.14.71 smrat::parseformula Namespace Reference	207
0.14.72 smrat::parser Namespace Reference	207
0.14.72.1 Typedef Documentation	209
0.14.72.2 Enumeration Type Documentation	209
0.14.72.3 Function Documentation	209
0.14.73 smrat::parser::arithmetic Namespace Reference	210
0.14.73.1 Typedef Documentation	210
0.14.73.2 Function Documentation	211
0.14.74 smrat::parser::conversion Namespace Reference	211
0.14.75 smrat::parser::core Namespace Reference	211
0.14.75.1 Function Documentation	211
0.14.76 smrat::parser::types Namespace Reference	212
0.14.76.1 Typedef Documentation	212
0.14.77 smrat::parser::uninterpreted Namespace Reference	213
0.14.77.1 Function Documentation	214
0.14.78 smrat::preprocessor Namespace Reference	214
0.14.79 smrat::qe Namespace Reference	214
0.14.79.1 Typedef Documentation	214
0.14.79.2 Enumeration Type Documentation	214
0.14.79.3 Function Documentation	215
0.14.80 smrat::qe::cad Namespace Reference	215
0.14.80.1 Function Documentation	215
0.14.81 smrat::qe::fm Namespace Reference	216
0.14.81.1 Function Documentation	216
0.14.82 smrat::resource Namespace Reference	216
0.14.82.1 Function Documentation	216
0.14.83 smrat::sat Namespace Reference	217
0.14.84 smrat::sat::detail Namespace Reference	217
0.14.84.1 Function Documentation	217
0.14.85 smrat::settings Namespace Reference	217
0.14.86 smrat::statistics Namespace Reference	217
0.14.86.1 Function Documentation	218
0.14.87 smrat::subtropical Namespace Reference	218
0.14.87.1 Detailed Description	218
0.14.87.2 Enumeration Type Documentation	218

0.14.87.3 Function Documentation	219
0.14.88 smrat::uf_impl Namespace Reference	220
0.14.89 smrat::unsatcore Namespace Reference	220
0.14.89.1 Detailed Description	220
0.14.90 smrat::validation Namespace Reference	220
0.14.90.1 Enumeration Type Documentation	221
0.14.90.2 Function Documentation	221
0.14.91 smrat::vb Namespace Reference	222
0.14.91.1 Function Documentation	222
0.14.92 smrat::vs Namespace Reference	222
0.14.92.1 Typedef Documentation	225
0.14.92.2 Function Documentation	225
0.15 Data Structure Documentation	235
0.15.1 smrat::AbstractModuleFactory Struct Reference	235
0.15.1.1 Constructor & Destructor Documentation	235
0.15.1.2 Member Function Documentation	235
0.15.2 smrat::parser::AbstractTheory Struct Reference	235
0.15.2.1 Detailed Description	236
0.15.2.2 Constructor & Destructor Documentation	236
0.15.2.3 Member Function Documentation	237
0.15.2.4 Field Documentation	238
0.15.3 smrat::cad::sample_compare::absvalue Struct Reference	238
0.15.4 smrat::analyzer::AnalysisSettings Struct Reference	238
0.15.4.1 Field Documentation	238
0.15.5 smrat::analyzer::AnalyzerStatistics Struct Reference	239
0.15.5.1 Member Function Documentation	239
0.15.6 smrat::AnnotatedBVTerm Class Reference	239
0.15.6.1 Constructor & Destructor Documentation	239
0.15.6.2 Member Function Documentation	240
0.15.6.3 Friends And Related Function Documentation	240
0.15.7 smrat::cadcells::representation::approximation::ApxCriteria Class Reference	240
0.15.7.1 Member Function Documentation	240
0.15.8 smrat::cadcells::representation::approximation::ApxSettings Struct Reference	241
0.15.8.1 Field Documentation	241
0.15.9 benchmax::slurm::ArchiveProperties Struct Reference	243
0.15.9.1 Detailed Description	243
0.15.9.2 Field Documentation	243
0.15.10 smrat::parser::ArithmeticTheory Struct Reference	244
0.15.10.1 Detailed Description	244
0.15.10.2 Constructor & Destructor Documentation	244
0.15.10.3 Member Function Documentation	245
0.15.10.4 Field Documentation	246

0.15.11 smrat::parser::types::ArithmeticTheory Struct Reference	246
0.15.11.1 Detailed Description	246
0.15.11.2 Member Typedef Documentation	247
0.15.12 smrat::execution::Assertion Struct Reference	247
0.15.12.1 Field Documentation	247
0.15.13 smrat::cad::preprocessor::AssignmentCollector Class Reference	247
0.15.13.1 Member Typedef Documentation	247
0.15.13.2 Constructor & Destructor Documentation	248
0.15.13.3 Member Function Documentation	248
0.15.14 smrat::mcsat::arithmetic::AssignmentFinder Struct Reference	248
0.15.14.1 Member Function Documentation	248
0.15.15 smrat::mcsat::smtaf::AssignmentFinder< Settings > Struct Template Reference	249
0.15.15.1 Member Function Documentation	249
0.15.16 smrat::mcsat::arithmetic::AssignmentFinder_ctx Class Reference	249
0.15.16.1 Constructor & Destructor Documentation	249
0.15.16.2 Member Function Documentation	249
0.15.17 smrat::mcsat::arithmetic::AssignmentFinder_detail Class Reference	250
0.15.17.1 Constructor & Destructor Documentation	250
0.15.17.2 Member Function Documentation	250
0.15.18 smrat::mcsat::smtaf::AssignmentFinder_SMT Class Reference	251
0.15.18.1 Constructor & Destructor Documentation	251
0.15.18.2 Member Function Documentation	251
0.15.19 smrat::cadcells::datastructures::detail::AssignmentProperties Struct Reference	251
0.15.19.1 Field Documentation	251
0.15.20 smrat::parser::Attribute Class Reference	252
0.15.20.1 Detailed Description	252
0.15.20.2 Member Typedef Documentation	252
0.15.20.3 Constructor & Destructor Documentation	252
0.15.20.4 Member Function Documentation	253
0.15.20.5 Field Documentation	253
0.15.21 smrat::parser::AttributeParser Struct Reference	253
0.15.21.1 Constructor & Destructor Documentation	253
0.15.21.2 Field Documentation	253
0.15.22 smrat::parser::AttributeValueParser Struct Reference	254
0.15.22.1 Member Typedef Documentation	254
0.15.22.2 Constructor & Destructor Documentation	254
0.15.22.3 Field Documentation	254
0.15.23 smrat::AxiomFactory Class Reference	254
0.15.23.1 Member Enumeration Documentation	255
0.15.23.2 Member Function Documentation	255
0.15.24 benchmax::Backend Class Reference	255
0.15.24.1 Detailed Description	256

0.15.24.2 Constructor & Destructor Documentation	256
0.15.24.3 Member Function Documentation	256
0.15.24.4 Field Documentation	257
0.15.25 smrat::Backend< Settings > Class Template Reference	257
0.15.25.1 Constructor & Destructor Documentation	258
0.15.25.2 Member Function Documentation	258
0.15.26 smrat::BackendLink Class Reference	260
0.15.26.1 Constructor & Destructor Documentation	260
0.15.26.2 Member Function Documentation	260
0.15.27 smrat::BackendSynchronisation Class Reference	261
0.15.27.1 Constructor & Destructor Documentation	261
0.15.27.2 Member Function Documentation	261
0.15.28 smrat::Backtrackable< UnionFind > Struct Template Reference	261
0.15.28.1 Member Typedef Documentation	262
0.15.28.2 Constructor & Destructor Documentation	262
0.15.28.3 Member Function Documentation	262
0.15.28.4 Field Documentation	263
0.15.29 smrat::cadcells::datastructures::BaseDerivation< Properties > Class Template Reference	263
0.15.29.1 Detailed Description	263
0.15.29.2 Constructor & Destructor Documentation	264
0.15.29.3 Member Function Documentation	264
0.15.29.4 Friends And Related Function Documentation	265
0.15.30 smrat::cad::BaseProjection< Settings > Class Template Reference	265
0.15.30.1 Member Typedef Documentation	266
0.15.30.2 Constructor & Destructor Documentation	266
0.15.30.3 Member Function Documentation	267
0.15.30.4 Field Documentation	270
0.15.31 smrat::cad::BaseSettings Struct Reference	270
0.15.31.1 Field Documentation	271
0.15.32 smrat::expression::simplifier::BaseSimplifier Struct Reference	271
0.15.32.1 Member Function Documentation	272
0.15.33 smrat::cad::Origin::BaseType Struct Reference	273
0.15.33.1 Constructor & Destructor Documentation	273
0.15.33.2 Member Function Documentation	274
0.15.33.3 Friends And Related Function Documentation	274
0.15.33.4 Field Documentation	274
0.15.34 smrat::mcsat::onecell::BCApproximationSettings Struct Reference	275
0.15.34.1 Field Documentation	275
0.15.35 smrat::mcsat::onecell::BCFilteredAllSelectiveSettings Struct Reference	275
0.15.35.1 Field Documentation	275
0.15.36 smrat::mcsat::onecell::BCFilteredAllSettings Struct Reference	276
0.15.36.1 Field Documentation	276

0.15.37 smrat::mcsat::onecell::BCFilteredBoundsSettings Struct Reference	276
0.15.37.1 Field Documentation	276
0.15.38 smrat::mcsat::onecell::BCFilteredSamplesSettings Struct Reference	276
0.15.38.1 Field Documentation	277
0.15.39 smrat::mcsat::onecell::BCFilteredSettings Struct Reference	277
0.15.39.1 Field Documentation	277
0.15.40 smrat::mcsat::onecell::BCSettings Struct Reference	277
0.15.40.1 Field Documentation	278
0.15.41 smrat::BEModule< Settings > Class Template Reference	278
0.15.41.1 Member Typedef Documentation	283
0.15.41.2 Member Enumeration Documentation	283
0.15.41.3 Constructor & Destructor Documentation	283
0.15.41.4 Member Function Documentation	283
0.15.41.5 Field Documentation	303
0.15.42 benchmax::BenchmarkResult Struct Reference	305
0.15.42.1 Detailed Description	306
0.15.42.2 Member Function Documentation	306
0.15.42.3 Field Documentation	306
0.15.43 benchmax::BenchmarkSet Class Reference	307
0.15.43.1 Detailed Description	307
0.15.43.2 Member Function Documentation	307
0.15.44 benchmax::settings::BenchmarkSettings Struct Reference	308
0.15.44.1 Detailed Description	308
0.15.44.2 Field Documentation	308
0.15.45 smrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName > Class Template Reference	309
0.15.45.1 Detailed Description	309
0.15.45.2 Member Typedef Documentation	309
0.15.45.3 Member Function Documentation	310
0.15.46 smrat::expression::BinaryExpression Struct Reference	311
0.15.46.1 Constructor & Destructor Documentation	311
0.15.46.2 Field Documentation	311
0.15.47 smrat::datastructures::BinaryHeap< T > Class Template Reference	312
0.15.47.1 Constructor & Destructor Documentation	312
0.15.47.2 Member Function Documentation	312
0.15.48 smrat::parser::BinaryParser Struct Reference	312
0.15.48.1 Detailed Description	313
0.15.48.2 Member Typedef Documentation	313
0.15.48.3 Constructor & Destructor Documentation	313
0.15.48.4 Member Function Documentation	313
0.15.48.5 Field Documentation	313
0.15.49 smrat::parser::BitvectorTheory Struct Reference	313

0.15.49.1 Detailed Description	314
0.15.49.2 Member Typedef Documentation	314
0.15.49.3 Constructor & Destructor Documentation	314
0.15.49.4 Member Function Documentation	315
0.15.49.5 Field Documentation	316
0.15.50 smtrat::parser::types::BitvectorTheory Struct Reference	316
0.15.50.1 Detailed Description	316
0.15.50.2 Member Typedef Documentation	317
0.15.51 smtrat::BlastedConstr Class Reference	317
0.15.51.1 Constructor & Destructor Documentation	317
0.15.51.2 Member Function Documentation	318
0.15.51.3 Friends And Related Function Documentation	318
0.15.52 smtrat::BlastedPoly Class Reference	318
0.15.52.1 Constructor & Destructor Documentation	318
0.15.52.2 Member Function Documentation	319
0.15.52.3 Friends And Related Function Documentation	319
0.15.53 smtrat::mcsat::Bookkeeping Class Reference	319
0.15.53.1 Detailed Description	320
0.15.53.2 Member Function Documentation	320
0.15.54 smtrat::parser::BooleanEncodingTheory Struct Reference	320
0.15.54.1 Detailed Description	321
0.15.54.2 Constructor & Destructor Documentation	321
0.15.54.3 Member Function Documentation	321
0.15.54.4 Field Documentation	323
0.15.55 Minisat::BoolOption Class Reference	323
0.15.55.1 Constructor & Destructor Documentation	323
0.15.55.2 Member Function Documentation	324
0.15.55.3 Field Documentation	324
0.15.56 smtrat::BoolUEQRewriter Struct Reference	324
0.15.56.1 Constructor & Destructor Documentation	325
0.15.56.2 Member Function Documentation	325
0.15.57 smtrat::cadcells::datastructures::Bound Class Reference	325
0.15.57.1 Detailed Description	325
0.15.57.2 Member Function Documentation	325
0.15.58 smtrat::lra::Bound< T1, T2 > Class Template Reference	326
0.15.58.1 Detailed Description	327
0.15.58.2 Member Typedef Documentation	327
0.15.58.3 Member Enumeration Documentation	328
0.15.58.4 Constructor & Destructor Documentation	328
0.15.58.5 Member Function Documentation	328
0.15.58.6 Friends And Related Function Documentation	335
0.15.59 smtrat::mcsat::fm::Bound Struct Reference	335

0.15.59.1 Constructor & Destructor Documentation	336
0.15.59.2 Friends And Related Function Documentation	336
0.15.59.3 Field Documentation	336
0.15.60 smtrat::vb::Bound< T > Class Template Reference	336
0.15.60.1 Detailed Description	337
0.15.60.2 Member Enumeration Documentation	337
0.15.60.3 Constructor & Destructor Documentation	338
0.15.60.4 Member Function Documentation	338
0.15.60.5 Friends And Related Function Documentation	341
0.15.61 smtrat::Branching Struct Reference	341
0.15.61.1 Detailed Description	341
0.15.61.2 Constructor & Destructor Documentation	341
0.15.61.3 Field Documentation	342
0.15.62 smtrat::BVAnnotation Class Reference	342
0.15.62.1 Constructor & Destructor Documentation	343
0.15.62.2 Member Function Documentation	343
0.15.62.3 Friends And Related Function Documentation	344
0.15.63 smtrat::BVDirectEncoder Class Reference	344
0.15.63.1 Constructor & Destructor Documentation	344
0.15.63.2 Member Function Documentation	344
0.15.64 smtrat::BVModule< Settings > Class Template Reference	344
0.15.64.1 Member Typedef Documentation	349
0.15.64.2 Member Enumeration Documentation	350
0.15.64.3 Constructor & Destructor Documentation	350
0.15.64.4 Member Function Documentation	350
0.15.64.5 Field Documentation	369
0.15.65 smtrat::by_address_hasher< IterType > Struct Template Reference	372
0.15.65.1 Detailed Description	372
0.15.65.2 Member Typedef Documentation	372
0.15.65.3 Member Function Documentation	372
0.15.66 smtrat::cad::CAD< Settings > Class Template Reference	372
0.15.66.1 Member Typedef Documentation	373
0.15.66.2 Constructor & Destructor Documentation	373
0.15.66.3 Member Function Documentation	373
0.15.66.4 Friends And Related Function Documentation	375
0.15.66.5 Field Documentation	375
0.15.67 smtrat::qe::cad::CAD< Settings > Class Template Reference	375
0.15.67.1 Member Typedef Documentation	375
0.15.67.2 Constructor & Destructor Documentation	376
0.15.67.3 Member Function Documentation	376
0.15.68 smtrat::cad::CADConstraints< BT > Class Template Reference	377
0.15.68.1 Member Typedef Documentation	378

0.15.68.2 Constructor & Destructor Documentation	378
0.15.68.3 Member Function Documentation	378
0.15.68.4 Friends And Related Function Documentation	380
0.15.68.5 Field Documentation	380
0.15.69 smrat::cad::CADCore< CH > Struct Template Reference	381
0.15.70 smrat::cad::CADCore< CoreHeuristic::BySample > Struct Reference	381
0.15.70.1 Member Function Documentation	381
0.15.71 smrat::cad::CADCore< CoreHeuristic::EnumerateAll > Struct Reference	381
0.15.71.1 Member Function Documentation	381
0.15.72 smrat::cad::CADCore< CoreHeuristic::Interleave > Struct Reference	382
0.15.72.1 Member Function Documentation	382
0.15.73 smrat::cad::CADCore< CoreHeuristic::PreferProjection > Struct Reference	382
0.15.73.1 Member Function Documentation	382
0.15.74 smrat::cad::CADCore< CoreHeuristic::PreferSampling > Struct Reference	383
0.15.74.1 Member Function Documentation	383
0.15.75 smrat::qe::cad::CADElimination Class Reference	383
0.15.75.1 Constructor & Destructor Documentation	383
0.15.75.2 Member Function Documentation	383
0.15.76 smrat::cad::CADPreprocessor Class Reference	383
0.15.76.1 Constructor & Destructor Documentation	384
0.15.76.2 Member Function Documentation	384
0.15.76.3 Friends And Related Function Documentation	385
0.15.77 smrat::cad::CADPreprocessorSettings Struct Reference	385
0.15.77.1 Member Function Documentation	385
0.15.77.2 Field Documentation	385
0.15.78 smrat::qe::cad::CADSettings Struct Reference	386
0.15.78.1 Field Documentation	386
0.15.79 smrat::CardinalityEncoder Class Reference	387
0.15.79.1 Constructor & Destructor Documentation	387
0.15.79.2 Member Function Documentation	387
0.15.79.3 Field Documentation	388
0.15.80 smrat::cadcells::representation::cell< H > Struct Template Reference	388
0.15.80.1 Detailed Description	388
0.15.80.2 Member Function Documentation	388
0.15.81 smrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL > Struct Reference	388
0.15.81.1 Member Function Documentation	389
0.15.82 smrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION > Struct Reference	389
0.15.82.1 Member Function Documentation	389
0.15.83 smrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_EW > Struct Reference	389
0.15.83.1 Member Function Documentation	389
0.15.84 smrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ > Struct Reference	389

0.15.84.1 Member Function Documentation	390
0.15.85 smrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS > Struct Reference	390
0.15.85.1 Member Function Documentation	390
0.15.86 smrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > Struct Reference	390
0.15.86.1 Member Function Documentation	390
0.15.87 smrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ > Struct Reference	390
0.15.87.1 Member Function Documentation	391
0.15.88 smrat::cadcells::operators::properties::cell_connected Struct Reference	391
0.15.88.1 Member Function Documentation	391
0.15.88.2 Field Documentation	391
0.15.89 smrat::cadcells::representation::approximation::CellApproximator Class Reference	391
0.15.89.1 Constructor & Destructor Documentation	392
0.15.89.2 Member Function Documentation	392
0.15.90 smrat::cadcells::datastructures::CellRepresentation< P > Struct Template Reference	392
0.15.90.1 Detailed Description	392
0.15.90.2 Constructor & Destructor Documentation	393
0.15.90.3 Field Documentation	393
0.15.91 delta::Checker Class Reference	393
0.15.91.1 Detailed Description	393
0.15.91.2 Constructor & Destructor Documentation	393
0.15.91.3 Member Function Documentation	394
0.15.92 benchmax::slurm::ChunkedSubmitfileProperties Struct Reference	394
0.15.92.1 Detailed Description	395
0.15.92.2 Field Documentation	395
0.15.93 Minisat::Clause Class Reference	396
0.15.93.1 Member Function Documentation	396
0.15.93.2 Friends And Related Function Documentation	398
0.15.93.3 Field Documentation	398
0.15.94 Minisat::ClauseAllocator Class Reference	398
0.15.94.1 Member Typedef Documentation	399
0.15.94.2 Member Enumeration Documentation	399
0.15.94.3 Constructor & Destructor Documentation	399
0.15.94.4 Member Function Documentation	400
0.15.94.5 Field Documentation	401
0.15.95 smrat::mcsat::ClauseChain Class Reference	401
0.15.95.1 Detailed Description	402
0.15.95.2 Member Typedef Documentation	402
0.15.95.3 Constructor & Destructor Documentation	402
0.15.95.4 Member Function Documentation	402
0.15.95.5 Friends And Related Function Documentation	403

0.15.96 smrat::sat::detail::ClauseChecker Struct Reference	403
0.15.96.1 Member Function Documentation	403
0.15.97 smrat::CMakeOptionPrinter Struct Reference	404
0.15.97.1 Field Documentation	404
0.15.98 Minisat::CMap< T > Class Template Reference	404
0.15.98.1 Member Function Documentation	404
0.15.99 smrat::CNFerModule Class Reference	405
0.15.99.1 Member Enumeration Documentation	411
0.15.99.2 Constructor & Destructor Documentation	412
0.15.99.3 Member Function Documentation	412
0.15.99.4 Field Documentation	432
0.15.100 smrat::CoCoAGBModule< Settings > Class Template Reference	434
0.15.100.1 Member Enumeration Documentation	439
0.15.100.2 Member Function Documentation	439
0.15.100.3 Field Documentation	459
0.15.101 smrat::CollectBoolsInUEQs Struct Reference	461
0.15.101.1 Constructor & Destructor Documentation	461
0.15.101.2 Member Function Documentation	462
0.15.101.3 Friends And Related Function Documentation	462
0.15.102 smrat::CollectionWithOrigins< Element, Origin > Class Template Reference	462
0.15.102.1 Member Typedef Documentation	462
0.15.102.2 Constructor & Destructor Documentation	463
0.15.102.3 Member Function Documentation	463
0.15.103 smrat::ICPModule< Settings >::comp Struct Reference	464
0.15.103.1 Detailed Description	464
0.15.103.2 Member Function Documentation	464
0.15.104 smrat::cadcells::datastructures::CompoundMax Struct Reference	464
0.15.104.1 Detailed Description	464
0.15.104.2 Member Function Documentation	464
0.15.104.3 Field Documentation	464
0.15.105 smrat::cadcells::datastructures::CompoundMin Struct Reference	464
0.15.105.1 Detailed Description	465
0.15.105.2 Member Function Documentation	465
0.15.105.3 Field Documentation	465
0.15.106 smrat::vs::Condition Class Reference	465
0.15.106.1 Constructor & Destructor Documentation	466
0.15.106.2 Member Function Documentation	466
0.15.106.3 Friends And Related Function Documentation	468
0.15.107 benchmax::CondorBackend Class Reference	468
0.15.107.1 Detailed Description	468
0.15.107.2 Member Function Documentation	469
0.15.107.3 Field Documentation	470

0.15.108 smrat::mcsat::fm::ConflictGenerator< Comparator > Struct Template Reference	470
0.15.108.1 Constructor & Destructor Documentation	470
0.15.108.2 Member Function Documentation	470
0.15.109 smrat::cad::ConflictGraph Class Reference	470
0.15.109.1 Detailed Description	471
0.15.109.2 Constructor & Destructor Documentation	471
0.15.109.3 Member Function Documentation	471
0.15.109.4 Friends And Related Function Documentation	472
0.15.110 smrat::parser::Theories::ConstantAdder Struct Reference	472
0.15.110.1 Detailed Description	473
0.15.110.2 Constructor & Destructor Documentation	473
0.15.110.3 Member Function Documentation	473
0.15.110.4 Field Documentation	473
0.15.111 smrat::cad::CADConstraints< BT >::ConstraintComparator Struct Reference	473
0.15.111.1 Member Function Documentation	473
0.15.112 smrat::cad::preprocessor::ConstraintUpdate Struct Reference	473
0.15.112.1 Field Documentation	474
0.15.113 smrat::ConstrTree Class Reference	474
0.15.113.1 Constructor & Destructor Documentation	474
0.15.113.2 Member Function Documentation	474
0.15.114 delta::Consumer Class Reference	475
0.15.114.1 Detailed Description	475
0.15.114.2 Constructor & Destructor Documentation	475
0.15.114.3 Member Function Documentation	475
0.15.115 smrat::LRAModule< Settings >::Context Struct Reference	476
0.15.115.1 Detailed Description	477
0.15.115.2 Constructor & Destructor Documentation	477
0.15.115.3 Field Documentation	477
0.15.116 smrat::icp::ContractionCandidate Class Reference	477
0.15.116.1 Constructor & Destructor Documentation	478
0.15.116.2 Member Function Documentation	479
0.15.117 smrat::icp::contractionCandidateComp Struct Reference	482
0.15.117.1 Member Function Documentation	482
0.15.118 smrat::icp::ContractionCandidateManager Class Reference	482
0.15.118.1 Constructor & Destructor Documentation	482
0.15.118.2 Member Function Documentation	482
0.15.119 smrat::parser::conversion::Converter< To > Struct Template Reference	484
0.15.119.1 Member Function Documentation	484
0.15.120 smrat::parser::conversion::Converter< FormulaT > Struct Reference	484
0.15.120.1 Member Function Documentation	484
0.15.121 smrat::parser::conversion::Converter< Poly > Struct Reference	485
0.15.121.1 Member Function Documentation	485

0.15.122 smrat::parser::conversion::Converter< types::BVTerm > Struct Reference	485
0.15.122.1 Member Function Documentation	486
0.15.123 benchmax::settings::CoreSettings Struct Reference	486
0.15.123.1 Detailed Description	487
0.15.123.2 Field Documentation	487
0.15.124 smrat::settings::CoreSettings Struct Reference	487
0.15.124.1 Field Documentation	487
0.15.125 smrat::parser::CoreTheory Struct Reference	488
0.15.125.1 Detailed Description	489
0.15.125.2 Member Typedef Documentation	489
0.15.125.3 Constructor & Destructor Documentation	489
0.15.125.4 Member Function Documentation	489
0.15.125.5 Field Documentation	490
0.15.126 smrat::parser::types::CoreTheory Struct Reference	490
0.15.126.1 Detailed Description	491
0.15.126.2 Member Typedef Documentation	491
0.15.127 smrat::cadcells::representation::covering< H > Struct Template Reference	491
0.15.127.1 Member Function Documentation	491
0.15.128 smrat::mcsat::arithmetic::Covering Class Reference	491
0.15.128.1 Detailed Description	492
0.15.128.2 Constructor & Destructor Documentation	492
0.15.128.3 Member Function Documentation	492
0.15.128.4 Friends And Related Function Documentation	492
0.15.129 smrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_↔ COVERING > Struct Reference	493
0.15.129.1 Member Function Documentation	493
0.15.130 smrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_↔ COVERING_EW > Struct Reference	493
0.15.130.1 Member Function Documentation	493
0.15.131 smrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING > Struct Reference	493
0.15.131.1 Member Function Documentation	493
0.15.132 smrat::cadcells::datastructures::CoveringDescription Class Reference	494
0.15.132.1 Detailed Description	494
0.15.132.2 Member Function Documentation	494
0.15.133 smrat::cadcells::datastructures::CoveringRepresentation< P > Struct Template Reference	494
0.15.133.1 Detailed Description	495
0.15.133.2 Member Function Documentation	495
0.15.133.3 Field Documentation	495
0.15.134 smrat::mcsat::onecellcad::recursive::CoverNullification Struct Reference	496
0.15.134.1 Field Documentation	496
0.15.135 smrat::CSplitModule< Settings > Class Template Reference	496
0.15.135.1 Member Typedef Documentation	501

0.15.135.2 Member Enumeration Documentation	501
0.15.135.3 Constructor & Destructor Documentation	501
0.15.135.4 Member Function Documentation	501
0.15.135.5 Field Documentation	520
0.15.136 smrat::CubeLIAModule< Settings > Class Template Reference	523
0.15.136.1 Member Typedef Documentation	528
0.15.136.2 Member Enumeration Documentation	528
0.15.136.3 Constructor & Destructor Documentation	528
0.15.136.4 Member Function Documentation	528
0.15.136.5 Field Documentation	547
0.15.137 smrat::CurryModule< Settings > Class Template Reference	550
0.15.137.1 Member Typedef Documentation	555
0.15.137.2 Member Enumeration Documentation	555
0.15.137.3 Constructor & Destructor Documentation	555
0.15.137.4 Member Function Documentation	555
0.15.137.5 Field Documentation	573
0.15.138 smrat::CycleEnumerator< FHG, Collector > Struct Template Reference	576
0.15.138.1 Detailed Description	576
0.15.138.2 Constructor & Destructor Documentation	576
0.15.138.3 Member Function Documentation	576
0.15.139 benchmax::Database Class Reference	576
0.15.139.1 Detailed Description	577
0.15.139.2 Member Typedef Documentation	577
0.15.139.3 Member Function Documentation	577
0.15.140 benchmax::DBAL Class Reference	578
0.15.140.1 Member Typedef Documentation	579
0.15.140.2 Member Function Documentation	579
0.15.141 smrat::parser::DecimalParser Struct Reference	582
0.15.141.1 Detailed Description	582
0.15.142 Minisat::DeepEqual< K > Struct Template Reference	582
0.15.142.1 Member Function Documentation	582
0.15.143 Minisat::DeepHash< K > Struct Template Reference	582
0.15.143.1 Member Function Documentation	582
0.15.144 smrat::mcsat::fm::DefaultComparator Struct Reference	582
0.15.144.1 Detailed Description	582
0.15.144.2 Member Function Documentation	583
0.15.144.3 Field Documentation	583
0.15.145 smrat::mcsat::fm::DefaultSettings Struct Reference	583
0.15.145.1 Field Documentation	583
0.15.146 smrat::mcsat::smtaf::DefaultSettings Struct Reference	583
0.15.146.1 Field Documentation	583
0.15.147 smrat::mcsat::vs::DefaultSettings Struct Reference	583

0.15.147.1 Field Documentation	584
0.15.148 smtrat::cad::projection_compare::degree Struct Reference	584
0.15.149 smtrat::mcsat::oncellcad::recursive::DegreeAscending Struct Reference	584
0.15.149.1 Field Documentation	584
0.15.150 smtrat::analyzer::DegreeCollector Struct Reference	584
0.15.150.1 Member Function Documentation	584
0.15.150.2 Field Documentation	585
0.15.151 smtrat::mcsat::oncellcad::recursive::DegreeDescending Struct Reference	585
0.15.151.1 Field Documentation	585
0.15.152 smtrat::cadcells::datastructures::DelineatedDerivation< Properties > Class Template Reference	585
0.15.152.1 Detailed Description	586
0.15.152.2 Constructor & Destructor Documentation	586
0.15.152.3 Member Function Documentation	586
0.15.153 smtrat::cadcells::datastructures::Delineation Class Reference	588
0.15.153.1 Detailed Description	588
0.15.153.2 Constructor & Destructor Documentation	588
0.15.153.3 Member Function Documentation	588
0.15.153.4 Friends And Related Function Documentation	589
0.15.154 smtrat::cadcells::datastructures::DelineationInterval Class Reference	589
0.15.154.1 Detailed Description	589
0.15.154.2 Member Function Documentation	589
0.15.154.3 Friends And Related Function Documentation	590
0.15.155 smtrat::mcsat::icp::Dependencies Class Reference	590
0.15.155.1 Member Function Documentation	590
0.15.156 smtrat::cadcells::datastructures::DerivationRef< Properties > Class Template Reference	591
0.15.156.1 Detailed Description	591
0.15.156.2 Constructor & Destructor Documentation	592
0.15.156.3 Member Function Documentation	592
0.15.156.4 Friends And Related Function Documentation	593
0.15.157 smtrat::mcsat::oncellcad::recursive::DontCoverNullification Struct Reference	594
0.15.157.1 Field Documentation	594
0.15.158 smtrat::cad::debug::DotSubgraph Struct Reference	594
0.15.158.1 Constructor & Destructor Documentation	594
0.15.158.2 Member Function Documentation	594
0.15.158.3 Friends And Related Function Documentation	594
0.15.158.4 Field Documentation	594
0.15.159 Minisat::DoubleOption Class Reference	595
0.15.159.1 Constructor & Destructor Documentation	595
0.15.159.2 Member Function Documentation	595
0.15.159.3 Field Documentation	596
0.15.160 Minisat::DoubleRange Struct Reference	596

0.15.160.1 Constructor & Destructor Documentation	596
0.15.160.2 Field Documentation	597
0.15.161 smrat::expression::simplifier::DuplicateSimplifier Struct Reference	597
0.15.161.1 Member Function Documentation	597
0.15.162 smrat::DynamicPriorityQueue< T, Compare > Class Template Reference	598
0.15.162.1 Member Typedef Documentation	599
0.15.162.2 Constructor & Destructor Documentation	599
0.15.162.3 Member Function Documentation	599
0.15.162.4 Field Documentation	600
0.15.163 smrat::dynarray< T > Class Template Reference	600
0.15.163.1 Detailed Description	601
0.15.163.2 Member Typedef Documentation	601
0.15.163.3 Constructor & Destructor Documentation	602
0.15.163.4 Member Function Documentation	602
0.15.164 smrat::dynarray_allocator< T > Class Template Reference	604
0.15.164.1 Friends And Related Function Documentation	604
0.15.165 smrat::datastructures::DynHeap< KeyType, Compare, WeightType > Class Template Reference	604
0.15.165.1 Constructor & Destructor Documentation	605
0.15.165.2 Member Function Documentation	605
0.15.166 smrat::cad::ProjectionGlobalInformation::ECData Struct Reference	606
0.15.166.1 Field Documentation	606
0.15.167 smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge Struct Reference	606
0.15.167.1 Detailed Description	606
0.15.167.2 Constructor & Destructor Documentation	607
0.15.167.3 Member Function Documentation	607
0.15.167.4 Field Documentation	607
0.15.168 smrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType > Struct Template Reference	608
0.15.168.1 Detailed Description	608
0.15.168.2 Member Typedef Documentation	608
0.15.168.3 Constructor & Destructor Documentation	609
0.15.168.4 Member Function Documentation	609
0.15.168.5 Friends And Related Function Documentation	610
0.15.169 smrat::ElementWithOrigins< Element, Origin > Class Template Reference	610
0.15.169.1 Constructor & Destructor Documentation	611
0.15.169.2 Member Function Documentation	611
0.15.170 smrat::EMModule< Settings > Class Template Reference	611
0.15.170.1 Member Typedef Documentation	616
0.15.170.2 Member Enumeration Documentation	616
0.15.170.3 Constructor & Destructor Documentation	617
0.15.170.4 Member Function Documentation	617

0.15.170.5 Field Documentation	637
0.15.171 smtrat::subtropical::Encoding Class Reference	639
0.15.171.1 Member Function Documentation	639
0.15.172 smtrat::EQGraph< VertexName > Struct Template Reference	640
0.15.172.1 Member Typedef Documentation	640
0.15.172.2 Member Function Documentation	641
0.15.172.3 Field Documentation	642
0.15.173 smtrat::EQModule< Settings > Class Template Reference	642
0.15.173.1 Member Typedef Documentation	647
0.15.173.2 Member Enumeration Documentation	647
0.15.173.3 Constructor & Destructor Documentation	647
0.15.173.4 Member Function Documentation	648
0.15.173.5 Field Documentation	666
0.15.174 smtrat::EQPreprocessingModule< Settings > Class Template Reference	668
0.15.174.1 Member Typedef Documentation	673
0.15.174.2 Member Enumeration Documentation	674
0.15.174.3 Constructor & Destructor Documentation	674
0.15.174.4 Member Function Documentation	674
0.15.174.5 Field Documentation	693
0.15.175 Minisat::Equal< K > Struct Template Reference	696
0.15.175.1 Member Function Documentation	696
0.15.176 delta::ErrorHandler Struct Reference	696
0.15.176.1 Member Function Documentation	696
0.15.177 smtrat::parser::ErrorHandler Struct Reference	696
0.15.177.1 Member Function Documentation	697
0.15.178 smtrat::ESModule< Settings > Class Template Reference	697
0.15.178.1 Member Typedef Documentation	702
0.15.178.2 Member Enumeration Documentation	702
0.15.178.3 Constructor & Destructor Documentation	702
0.15.178.4 Member Function Documentation	702
0.15.178.5 Field Documentation	722
0.15.179 smtrat::uf_impl::estimator< IteratorType, IsRandomAccess > Struct Template Reference	724
0.15.179.1 Detailed Description	725
0.15.179.2 Member Function Documentation	725
0.15.180 smtrat::uf_impl::estimator< IteratorType, false > Struct Template Reference	725
0.15.180.1 Member Function Documentation	725
0.15.181 smtrat::ExactlyOneCommanderEncoder Class Reference	725
0.15.181.1 Constructor & Destructor Documentation	726
0.15.181.2 Member Function Documentation	726
0.15.181.3 Field Documentation	726
0.15.182 smtrat::execution::ExecutionState Class Reference	727
0.15.182.1 Constructor & Destructor Documentation	727

0.15.182.2 Member Function Documentation	727
0.15.183 smtrat::Executor< Strategy > Class Template Reference	730
0.15.183.1 Member Typedef Documentation	731
0.15.183.2 Constructor & Destructor Documentation	731
0.15.183.3 Member Function Documentation	732
0.15.183.4 Field Documentation	736
0.15.184 smtrat::mcsat::fm::Explanation< Settings > Struct Template Reference	736
0.15.184.1 Member Function Documentation	736
0.15.185 smtrat::mcsat::icp::Explanation Struct Reference	737
0.15.185.1 Member Function Documentation	737
0.15.186 smtrat::mcsat::nlsat::Explanation Struct Reference	737
0.15.186.1 Member Function Documentation	737
0.15.187 smtrat::mcsat::onecell::Explanation Struct Reference	738
0.15.187.1 Member Function Documentation	738
0.15.188 smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 > Struct Template Reference	738
0.15.188.1 Member Function Documentation	738
0.15.189 smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 > Struct Template Reference	738
0.15.189.1 Member Function Documentation	738
0.15.190 smtrat::mcsat::vs::Explanation Struct Reference	739
0.15.190.1 Member Function Documentation	739
0.15.191 smtrat::mcsat::nlsat::ExplanationGenerator Class Reference	739
0.15.191.1 Constructor & Destructor Documentation	739
0.15.191.2 Member Function Documentation	739
0.15.192 smtrat::mcsat::vs::ExplanationGenerator< Settings > Class Template Reference	740
0.15.192.1 Constructor & Destructor Documentation	740
0.15.192.2 Member Function Documentation	740
0.15.193 smtrat::expression::Expression Class Reference	740
0.15.193.1 Constructor & Destructor Documentation	741
0.15.193.2 Member Function Documentation	742
0.15.193.3 Friends And Related Function Documentation	743
0.15.194 smtrat::expression::ExpressionContent Struct Reference	743
0.15.194.1 Member Typedef Documentation	744
0.15.194.2 Constructor & Destructor Documentation	744
0.15.194.3 Friends And Related Function Documentation	744
0.15.194.4 Field Documentation	744
0.15.195 smtrat::expression::ExpressionConverter Struct Reference	744
0.15.195.1 Member Function Documentation	745
0.15.195.2 Field Documentation	745
0.15.196 smtrat::expression::ExpressionModifier Class Reference	745
0.15.196.1 Member Typedef Documentation	746
0.15.196.2 Member Function Documentation	746

0.15.197 smrat::expression::ExpressionPool Class Reference	747
0.15.197.1 Constructor & Destructor Documentation	747
0.15.197.2 Member Function Documentation	747
0.15.197.3 Friends And Related Function Documentation	748
0.15.198 smrat::parser::ParserState::ExpressionScope Struct Reference	748
0.15.198.1 Constructor & Destructor Documentation	748
0.15.198.2 Member Function Documentation	749
0.15.199 smrat::expression::ExpressionTypeChecker< T > Struct Template Reference	749
0.15.199.1 Member Function Documentation	749
0.15.200 smrat::expression::ExpressionVisitor Class Reference	749
0.15.200.1 Member Typedef Documentation	749
0.15.200.2 Member Function Documentation	750
0.15.201 smrat::mcsat::FastParallelExplanation< Backends > Struct Template Reference	750
0.15.201.1 Detailed Description	750
0.15.202 smrat::mcsat::variableordering::detail::FeatureCollector< Objects > Struct Template Reference	751
0.15.202.1 Detailed Description	751
0.15.202.2 Member Typedef Documentation	751
0.15.202.3 Member Function Documentation	751
0.15.202.4 Field Documentation	752
0.15.203 smrat::fixedsize_allocator< T > Class Template Reference	752
0.15.203.1 Detailed Description	752
0.15.203.2 Constructor & Destructor Documentation	752
0.15.203.3 Member Function Documentation	753
0.15.204 smrat::fixedsize_allocator< void > Class Reference	753
0.15.204.1 Detailed Description	753
0.15.204.2 Field Documentation	753
0.15.205 smrat::impl::fixedsize_allocator_freeLists Class Reference	753
0.15.205.1 Field Documentation	754
0.15.206 smrat::impl::fixedsize_allocator_Impl< T, SizeShift, LargeEnough, SmallEnough > Class Template Reference	754
0.15.207 smrat::impl::fixedsize_allocator_Impl< T, SizeShift, true, true > Class Template Reference	754
0.15.207.1 Member Typedef Documentation	754
0.15.207.2 Member Function Documentation	755
0.15.208 smrat::impl::fixedsize_allocator_size_helper< T, Size > Class Template Reference	756
0.15.209 smrat::fixedsize_freeList< ChunkSizeShift > Class Template Reference	756
0.15.209.1 Detailed Description	756
0.15.209.2 Constructor & Destructor Documentation	756
0.15.209.3 Member Function Documentation	756
0.15.210 smrat::impl::fixedsize_freeLists< SizeCurrent, SizeMax, LowEnough > Class Template Reference	757
0.15.210.1 Detailed Description	757
0.15.211 smrat::impl::fixedsize_freeLists< SizeCurrent, SizeMax, true > Class Template Reference	757

0.15.212 smrat::parser::FixedWidthConstant< T > Struct Template Reference	757
0.15.212.1 Detailed Description	757
0.15.212.2 Constructor & Destructor Documentation	757
0.15.212.3 Member Function Documentation	758
0.15.212.4 Field Documentation	758
0.15.213 smrat::parseformula::FormulaCollector Class Reference	758
0.15.213.1 Member Typedef Documentation	759
0.15.213.2 Member Function Documentation	759
0.15.213.3 Field Documentation	763
0.15.214 smrat::ICPModule< Settings >::formulaPtrComp Struct Reference	764
0.15.214.1 Member Function Documentation	764
0.15.215 smrat::FormulaWithOrigins Class Reference	764
0.15.215.1 Detailed Description	765
0.15.215.2 Constructor & Destructor Documentation	765
0.15.215.3 Member Function Documentation	765
0.15.215.4 Friends And Related Function Documentation	766
0.15.216 smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices > Class Template Reference	767
0.15.216.1 Detailed Description	768
0.15.216.2 Member Typedef Documentation	768
0.15.216.3 Member Function Documentation	768
0.15.216.4 Friends And Related Function Documentation	770
0.15.217 smrat::FouMoModule< Settings > Class Template Reference	770
0.15.217.1 Detailed Description	775
0.15.217.2 Member Typedef Documentation	775
0.15.217.3 Member Enumeration Documentation	775
0.15.217.4 Constructor & Destructor Documentation	775
0.15.217.5 Member Function Documentation	776
0.15.217.6 Field Documentation	794
0.15.218 smrat::qe::fm::FourierMotzkinQE Class Reference	796
0.15.218.1 Detailed Description	796
0.15.218.2 Member Typedef Documentation	796
0.15.218.3 Constructor & Destructor Documentation	796
0.15.218.4 Member Function Documentation	797
0.15.219 smrat::FPPModule< Settings > Class Template Reference	797
0.15.219.1 Member Typedef Documentation	802
0.15.219.2 Member Enumeration Documentation	802
0.15.219.3 Constructor & Destructor Documentation	802
0.15.219.4 Member Function Documentation	802
0.15.219.5 Field Documentation	821
0.15.220 smrat::freelist< T > Class Template Reference	823
0.15.220.1 Detailed Description	823

0.15.220.2 Constructor & Destructor Documentation	823
0.15.220.3 Member Function Documentation	824
0.15.221 smrat::mcsat::FullParallelExplanation< Backends > Struct Template Reference	824
0.15.221.1 Detailed Description	824
0.15.222 smrat::cad::FullSampleComparator< Iterator, Strategy > Struct Template Reference	825
0.15.223 smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > Struct Template Reference	825
0.15.224 smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > Struct Template Reference	825
0.15.225 smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > Struct Template Reference	825
0.15.226 smrat::parser::FunctionInstantiator Struct Reference	825
0.15.226.1 Constructor & Destructor Documentation	825
0.15.226.2 Member Function Documentation	825
0.15.227 smrat::GBModule< Settings > Class Template Reference	826
0.15.227.1 Detailed Description	832
0.15.227.2 Member Typedef Documentation	832
0.15.227.3 Member Enumeration Documentation	832
0.15.227.4 Constructor & Destructor Documentation	832
0.15.227.5 Member Function Documentation	833
0.15.227.6 Friends And Related Function Documentation	854
0.15.227.7 Field Documentation	854
0.15.228 smrat::GBModuleState< Settings > Class Template Reference	857
0.15.228.1 Detailed Description	858
0.15.228.2 Constructor & Destructor Documentation	858
0.15.228.3 Member Function Documentation	858
0.15.228.4 Field Documentation	858
0.15.229 smrat::GBPPModule< Settings > Class Template Reference	858
0.15.229.1 Member Typedef Documentation	863
0.15.229.2 Member Enumeration Documentation	863
0.15.229.3 Constructor & Destructor Documentation	864
0.15.229.4 Member Function Documentation	864
0.15.229.5 Field Documentation	884
0.15.230 Minisat::Hash< K > Struct Template Reference	886
0.15.230.1 Member Function Documentation	886
0.15.231 smrat::EQModule< Settings >::not_asserted_equality::hash Struct Reference	886
0.15.231.1 Member Function Documentation	886
0.15.232 std::hash< const smrat::expression::ExpressionContent * > Struct Reference	886
0.15.232.1 Member Function Documentation	887
0.15.233 std::hash< smrat::expression::BinaryExpression > Struct Reference	887
0.15.233.1 Member Function Documentation	887
0.15.234 std::hash< smrat::expression::Expression > Struct Reference	887
0.15.234.1 Member Function Documentation	887

0.15.235 std::hash< smrat::expression::ITEExpression > Struct Reference	887
0.15.235.1 Member Function Documentation	887
0.15.236 std::hash< smrat::expression::NaryExpression > Struct Reference	888
0.15.236.1 Member Function Documentation	888
0.15.237 std::hash< smrat::expression::QuantifierExpression > Struct Reference	888
0.15.237.1 Member Function Documentation	888
0.15.238 std::hash< smrat::expression::UnaryExpression > Struct Reference	888
0.15.238.1 Member Function Documentation	888
0.15.239 std::hash< smrat::lra::Variable< T1, T2 > > Struct Template Reference	888
0.15.239.1 Detailed Description	889
0.15.239.2 Member Function Documentation	889
0.15.240 std::hash< smrat::vs::Substitution > Struct Reference	889
0.15.240.1 Member Function Documentation	889
0.15.241 std::hash< std::vector< T > > Struct Template Reference	889
0.15.241.1 Member Function Documentation	889
0.15.242 std::hash_combiner Struct Reference	889
0.15.242.1 Constructor & Destructor Documentation	890
0.15.242.2 Member Function Documentation	890
0.15.242.3 Field Documentation	890
0.15.243 Minisat::Heap< Comp > Class Template Reference	890
0.15.243.1 Constructor & Destructor Documentation	891
0.15.243.2 Member Function Documentation	891
0.15.244 smrat::parser::HexadecimalParser Struct Reference	892
0.15.244.1 Detailed Description	892
0.15.244.2 Member Typedef Documentation	892
0.15.244.3 Constructor & Destructor Documentation	892
0.15.244.4 Member Function Documentation	892
0.15.244.5 Field Documentation	893
0.15.245 smrat::icp::HistoryNode Class Reference	893
0.15.245.1 Constructor & Destructor Documentation	894
0.15.245.2 Member Function Documentation	894
0.15.246 smrat::ICEModule< Settings > Class Template Reference	897
0.15.246.1 Member Typedef Documentation	902
0.15.246.2 Member Enumeration Documentation	902
0.15.246.3 Constructor & Destructor Documentation	902
0.15.246.4 Member Function Documentation	902
0.15.246.5 Field Documentation	921
0.15.247 smrat::ICPModule< Settings > Class Template Reference	923
0.15.247.1 Member Typedef Documentation	929
0.15.247.2 Member Enumeration Documentation	929
0.15.247.3 Constructor & Destructor Documentation	929
0.15.247.4 Member Function Documentation	929

0.15.247.5 Field Documentation	948
0.15.248 smtrat::icp::IcpVariable Class Reference	950
0.15.248.1 Constructor & Destructor Documentation	951
0.15.248.2 Member Function Documentation	951
0.15.248.3 Friends And Related Function Documentation	953
0.15.249 smtrat::icp::IcpVariableComp Struct Reference	954
0.15.249.1 Member Function Documentation	954
0.15.250 smtrat::parser::Identifier Struct Reference	954
0.15.250.1 Constructor & Destructor Documentation	954
0.15.250.2 Member Function Documentation	955
0.15.250.3 Field Documentation	955
0.15.251 smtrat::parser::IdentifierParser Struct Reference	955
0.15.251.1 Constructor & Destructor Documentation	955
0.15.251.2 Field Documentation	955
0.15.252 smtrat::cad::debug::IDSanitizer Struct Reference	956
0.15.252.1 Constructor & Destructor Documentation	956
0.15.252.2 Member Function Documentation	956
0.15.253 smtrat::mcsat::fm::IgnoreCoreSettings Struct Reference	956
0.15.253.1 Field Documentation	956
0.15.254 smtrat::IncWidthModule< Settings > Class Template Reference	956
0.15.254.1 Member Typedef Documentation	962
0.15.254.2 Member Enumeration Documentation	962
0.15.254.3 Constructor & Destructor Documentation	962
0.15.254.4 Member Function Documentation	962
0.15.254.5 Field Documentation	981
0.15.255 smtrat::parser::IndexedFunctionInstantiator Struct Reference	984
0.15.255.1 Constructor & Destructor Documentation	984
0.15.255.2 Member Function Documentation	984
0.15.256 smtrat::cadcells::datastructures::IndexedRoot Struct Reference	984
0.15.256.1 Detailed Description	985
0.15.256.2 Constructor & Destructor Documentation	985
0.15.256.3 Field Documentation	985
0.15.257 smtrat::cadcells::datastructures::IndexedRootOrdering Class Reference	985
0.15.257.1 Detailed Description	985
0.15.257.2 Member Function Documentation	986
0.15.258 smtrat::cadcells::datastructures::IndexedRootRelation Struct Reference	987
0.15.258.1 Detailed Description	987
0.15.258.2 Field Documentation	987
0.15.259 smtrat::InequalitiesTable< Settings > Class Template Reference	987
0.15.259.1 Detailed Description	988
0.15.259.2 Member Typedef Documentation	988
0.15.259.3 Constructor & Destructor Documentation	989

0.15.259.4 Member Function Documentation	989
0.15.259.5 Field Documentation	990
0.15.260 smrat::ira::Bound< T1, T2 >::Info Struct Reference	990
0.15.260.1 Detailed Description	991
0.15.260.2 Constructor & Destructor Documentation	991
0.15.260.3 Field Documentation	991
0.15.261 smrat::mcsat::InformationGetter Struct Reference	992
0.15.261.1 Field Documentation	992
0.15.262 smrat::parser::Instantiator< V, T > Struct Template Reference	993
0.15.262.1 Member Function Documentation	994
0.15.262.2 Field Documentation	994
0.15.263 smrat::parser::InstructionHandler Class Reference	995
0.15.263.1 Member Typedef Documentation	996
0.15.263.2 Constructor & Destructor Documentation	996
0.15.263.3 Member Function Documentation	996
0.15.263.4 Field Documentation	1000
0.15.264 Minisat::Int64Range Struct Reference	1001
0.15.264.1 Constructor & Destructor Documentation	1001
0.15.264.2 Field Documentation	1001
0.15.265 smrat::IntBlastModule< Settings > Class Template Reference	1001
0.15.265.1 Member Typedef Documentation	1006
0.15.265.2 Member Enumeration Documentation	1006
0.15.265.3 Constructor & Destructor Documentation	1006
0.15.265.4 Member Function Documentation	1007
0.15.265.5 Field Documentation	1025
0.15.266 smrat::IntEqModule< Settings > Class Template Reference	1027
0.15.266.1 Detailed Description	1032
0.15.266.2 Member Typedef Documentation	1032
0.15.266.3 Member Enumeration Documentation	1032
0.15.266.4 Constructor & Destructor Documentation	1033
0.15.266.5 Member Function Documentation	1033
0.15.266.6 Field Documentation	1052
0.15.267 smrat::mcsat::icp::IntervalPropagation Class Reference	1054
0.15.267.1 Constructor & Destructor Documentation	1054
0.15.267.2 Member Function Documentation	1054
0.15.268 Minisat::IntOption Class Reference	1054
0.15.268.1 Constructor & Destructor Documentation	1055
0.15.268.2 Member Function Documentation	1055
0.15.268.3 Field Documentation	1056
0.15.269 Minisat::IntRange Struct Reference	1056
0.15.269.1 Constructor & Destructor Documentation	1056
0.15.269.2 Field Documentation	1056

0.15.270 smrat::is_sample_outside< S > Struct Template Reference	1056
0.15.270.1 Member Function Documentation	1057
0.15.271 smrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT > Struct Reference	1057
0.15.271.1 Member Function Documentation	1057
0.15.272 smrat::is_variant< T > Struct Template Reference	1057
0.15.272.1 Detailed Description	1057
0.15.273 smrat::expression::ITEExpression Struct Reference	1057
0.15.273.1 Constructor & Destructor Documentation	1058
0.15.273.2 Field Documentation	1058
0.15.274 benchmax::RandomizationAdaptor< T >::iterator Struct Reference	1058
0.15.274.1 Constructor & Destructor Documentation	1058
0.15.274.2 Member Function Documentation	1059
0.15.275 smrat::ModuleInput::IteratorCompare Struct Reference	1059
0.15.275.1 Member Function Documentation	1059
0.15.276 benchmax::Jobs Class Reference	1059
0.15.276.1 Detailed Description	1060
0.15.276.2 Constructor & Destructor Documentation	1060
0.15.276.3 Member Function Documentation	1060
0.15.277 smrat::JunctorMerger Struct Reference	1060
0.15.277.1 Member Function Documentation	1060
0.15.278 smrat::parser::KeywordParser Struct Reference	1061
0.15.278.1 Detailed Description	1061
0.15.278.2 Constructor & Destructor Documentation	1061
0.15.278.3 Field Documentation	1061
0.15.279 Minisat::lbool Class Reference	1061
0.15.279.1 Constructor & Destructor Documentation	1062
0.15.279.2 Member Function Documentation	1062
0.15.279.3 Friends And Related Function Documentation	1062
0.15.280 smrat::mcsat::onecell::LDBFilteredAllSelectiveSettings Struct Reference	1062
0.15.280.1 Field Documentation	1063
0.15.281 smrat::mcsat::onecell::LDBSettings Struct Reference	1063
0.15.281.1 Field Documentation	1063
0.15.282 smrat::ira::Tableau< Settings, T1, T2 >::LearnedBound Struct Reference	1063
0.15.282.1 Constructor & Destructor Documentation	1064
0.15.282.2 Field Documentation	1064
0.15.283 smrat::Module::Lemma Struct Reference	1065
0.15.283.1 Constructor & Destructor Documentation	1065
0.15.283.2 Field Documentation	1065
0.15.284 Minisat::LessThan_default< T > Struct Template Reference	1066
0.15.284.1 Member Function Documentation	1066
0.15.285 smrat::cad::projection_compare::level Struct Reference	1066
0.15.286 smrat::cad::sample_compare::level Struct Reference	1066

0.15.287 smrat::cad::ProjectionLevelInformation::LevelInfo Struct Reference	1066
0.15.287.1 Member Function Documentation	1066
0.15.287.2 Field Documentation	1067
0.15.288 smrat::mcsat::oncellcad::levelwise::LevelwiseCAD Class Reference	1068
0.15.288.1 Member Function Documentation	1068
0.15.288.2 Field Documentation	1069
0.15.289 smrat::LevelWiseInformation< Settings > Class Template Reference	1070
0.15.289.1 Constructor & Destructor Documentation	1070
0.15.289.2 Member Function Documentation	1070
0.15.290 smrat::cad::LiftingTree< Settings > Class Template Reference	1072
0.15.290.1 Member Typedef Documentation	1073
0.15.290.2 Constructor & Destructor Documentation	1073
0.15.290.3 Member Function Documentation	1073
0.15.291 smrat::resource::Limiter Class Reference	1075
0.15.291.1 Member Function Documentation	1075
0.15.292 smrat::ICPModule< Settings >::linearVariable Struct Reference	1076
0.15.292.1 Field Documentation	1076
0.15.293 smrat::mcsat::ClauseChain::Link Struct Reference	1076
0.15.293.1 Constructor & Destructor Documentation	1077
0.15.293.2 Member Function Documentation	1077
0.15.293.3 Friends And Related Function Documentation	1077
0.15.294 Minisat::Lit Struct Reference	1077
0.15.294.1 Member Function Documentation	1078
0.15.294.2 Friends And Related Function Documentation	1078
0.15.294.3 Field Documentation	1078
0.15.295 benchmax::LocalBackend Class Reference	1078
0.15.295.1 Detailed Description	1079
0.15.295.2 Member Function Documentation	1079
0.15.295.3 Field Documentation	1080
0.15.296 smrat::LOG Class Reference	1080
0.15.296.1 Member Function Documentation	1080
0.15.296.2 Friends And Related Function Documentation	1081
0.15.296.3 Field Documentation	1081
0.15.297 smrat::parser::LogicParser Struct Reference	1081
0.15.297.1 Constructor & Destructor Documentation	1081
0.15.298 smrat::LongFormulaEncoder Class Reference	1081
0.15.298.1 Constructor & Destructor Documentation	1081
0.15.298.2 Member Function Documentation	1082
0.15.298.3 Field Documentation	1082
0.15.299 smrat::LRAbstractModule< Settings > Class Template Reference	1082
0.15.299.1 Detailed Description	1088
0.15.299.2 Member Typedef Documentation	1089

0.15.299.3 Member Enumeration Documentation	1090
0.15.299.4 Constructor & Destructor Documentation	1090
0.15.299.5 Member Function Documentation	1090
0.15.299.6 Field Documentation	1111
0.15.300 smrat::LVEModule< Settings > Class Template Reference	1114
0.15.300.1 Member Typedef Documentation	1119
0.15.300.2 Member Enumeration Documentation	1119
0.15.300.3 Constructor & Destructor Documentation	1119
0.15.300.4 Member Function Documentation	1119
0.15.300.5 Field Documentation	1138
0.15.301 smrat::Manager Class Reference	1140
0.15.301.1 Detailed Description	1142
0.15.301.2 Constructor & Destructor Documentation	1142
0.15.301.3 Member Function Documentation	1142
0.15.301.4 Friends And Related Function Documentation	1148
0.15.302 Minisat::Map< K, D, H, E > Class Template Reference	1148
0.15.302.1 Constructor & Destructor Documentation	1149
0.15.302.2 Member Function Documentation	1149
0.15.303 benchmax::MathSAT Class Reference	1150
0.15.303.1 Detailed Description	1151
0.15.303.2 Constructor & Destructor Documentation	1151
0.15.303.3 Member Function Documentation	1151
0.15.303.4 Field Documentation	1152
0.15.304 smrat::mcsat::fm::MaxSizeComparator Struct Reference	1153
0.15.304.1 Detailed Description	1153
0.15.304.2 Member Function Documentation	1153
0.15.304.3 Field Documentation	1153
0.15.305 smrat::MaxSMT< Solver, Strategy > Class Template Reference	1153
0.15.305.1 Constructor & Destructor Documentation	1153
0.15.305.2 Member Function Documentation	1153
0.15.306 smrat::maxsmt::MaxSMTBackend< Solver, Strategy > Class Template Reference	1154
0.15.307 smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL > Class Template Reference	1154
0.15.307.1 Constructor & Destructor Documentation	1154
0.15.307.2 Member Function Documentation	1154
0.15.308 smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH > Class Template Reference	1155
0.15.308.1 Constructor & Destructor Documentation	1155
0.15.308.2 Member Function Documentation	1155
0.15.309 smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 > Class Template Reference	1155
0.15.309.1 Constructor & Destructor Documentation	1155
0.15.309.2 Member Function Documentation	1155

0.15.310 smrat::MCBModule< Settings > Class Template Reference	1155
0.15.310.1 Member Typedef Documentation	1161
0.15.310.2 Member Enumeration Documentation	1161
0.15.310.3 Constructor & Destructor Documentation	1161
0.15.310.4 Member Function Documentation	1161
0.15.310.5 Field Documentation	1180
0.15.311 smrat::mcsat::MCSATBackend< Settings > Class Template Reference	1182
0.15.311.1 Member Function Documentation	1183
0.15.312 smrat::mcsat::MCSATMixin< Settings > Class Template Reference	1184
0.15.312.1 Constructor & Destructor Documentation	1185
0.15.312.2 Member Function Documentation	1185
0.15.312.3 Friends And Related Function Documentation	1189
0.15.313 smrat::expression::simplifier::MergeSimplifier Struct Reference	1190
0.15.313.1 Member Function Documentation	1190
0.15.314 benchmax::Minisat Class Reference	1191
0.15.314.1 Detailed Description	1192
0.15.314.2 Constructor & Destructor Documentation	1192
0.15.314.3 Member Function Documentation	1192
0.15.314.4 Field Documentation	1193
0.15.315 benchmax::Minisatp Class Reference	1194
0.15.315.1 Detailed Description	1194
0.15.315.2 Constructor & Destructor Documentation	1194
0.15.315.3 Member Function Documentation	1195
0.15.315.4 Field Documentation	1196
0.15.316 smrat::mcsat::fm::MinSizeComparator Struct Reference	1196
0.15.316.1 Detailed Description	1196
0.15.316.2 Member Function Documentation	1196
0.15.316.3 Field Documentation	1196
0.15.317 smrat::mcsat::fm::MinVarCountComparator Struct Reference	1197
0.15.317.1 Detailed Description	1197
0.15.317.2 Member Function Documentation	1197
0.15.317.3 Field Documentation	1197
0.15.318 smrat::cad::MISGeneration< heuristic > Class Template Reference	1197
0.15.318.1 Member Function Documentation	1197
0.15.319 smrat::MixedSignEncoder Class Reference	1198
0.15.319.1 Constructor & Destructor Documentation	1199
0.15.319.2 Member Function Documentation	1199
0.15.319.3 Field Documentation	1199
0.15.320 smrat::cad::ModelBasedProjection< incrementality, backtracking, Settings > Class Template Reference	1199
0.15.320.1 Member Typedef Documentation	1201
0.15.320.2 Member Function Documentation	1201

0.15.320.3 Field Documentation	1204
0.15.321 smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings > Class Template Reference	1205
0.15.321.1 Detailed Description	1206
0.15.321.2 Constructor & Destructor Documentation	1206
0.15.321.3 Member Function Documentation	1207
0.15.321.4 Friends And Related Function Documentation	1210
0.15.321.5 Field Documentation	1211
0.15.322 smrat::Module Class Reference	1211
0.15.322.1 Detailed Description	1216
0.15.322.2 Member Enumeration Documentation	1216
0.15.322.3 Constructor & Destructor Documentation	1216
0.15.322.4 Member Function Documentation	1217
0.15.322.5 Field Documentation	1236
0.15.323 smrat::ModuleFactory< Module > Struct Template Reference	1238
0.15.323.1 Constructor & Destructor Documentation	1238
0.15.323.2 Member Function Documentation	1238
0.15.324 smrat::ModuleInput Class Reference	1238
0.15.324.1 Detailed Description	1240
0.15.324.2 Member Typedef Documentation	1240
0.15.324.3 Constructor & Destructor Documentation	1240
0.15.324.4 Member Function Documentation	1240
0.15.324.5 Friends And Related Function Documentation	1244
0.15.325 smrat::settings::ModuleSettings Struct Reference	1244
0.15.325.1 Constructor & Destructor Documentation	1245
0.15.325.2 Member Function Documentation	1245
0.15.325.3 Field Documentation	1245
0.15.326 smrat::Module::ModuleStatistics Struct Reference	1245
0.15.326.1 Member Function Documentation	1245
0.15.327 smrat::ModuleWrapper< M > Class Template Reference	1246
0.15.327.1 Constructor & Destructor Documentation	1246
0.15.327.2 Member Function Documentation	1246
0.15.328 smrat::subtropical::Moment Struct Reference	1247
0.15.328.1 Detailed Description	1247
0.15.328.2 Constructor & Destructor Documentation	1247
0.15.328.3 Field Documentation	1247
0.15.329 smrat::MonomialMappingByVariablePool Class Reference	1247
0.15.329.1 Member Function Documentation	1248
0.15.329.2 Friends And Related Function Documentation	1248
0.15.330 smrat::expression::NaryExpression Struct Reference	1248
0.15.330.1 Constructor & Destructor Documentation	1249
0.15.330.2 Member Function Documentation	1249

0.15.330.3 Field Documentation	1249
0.15.331 smrat::expression::simplifier::NegationSimplifier Struct Reference	1249
0.15.331.1 Member Function Documentation	1249
0.15.332 smrat::NewCADModule< Settings > Class Template Reference	1251
0.15.332.1 Member Typedef Documentation	1256
0.15.332.2 Member Enumeration Documentation	1256
0.15.332.3 Constructor & Destructor Documentation	1256
0.15.332.4 Member Function Documentation	1256
0.15.332.5 Field Documentation	1275
0.15.333 smrat::NewCoveringModule< Settings > Class Template Reference	1277
0.15.333.1 Member Typedef Documentation	1282
0.15.333.2 Member Enumeration Documentation	1282
0.15.333.3 Constructor & Destructor Documentation	1282
0.15.333.4 Member Function Documentation	1283
0.15.333.5 Field Documentation	1302
0.15.334 smrat::NNFPreparation Struct Reference	1305
0.15.334.1 Member Function Documentation	1305
0.15.335 smrat::NNFRewriter Class Reference	1305
0.15.335.1 Constructor & Destructor Documentation	1306
0.15.335.2 Member Function Documentation	1306
0.15.336 benchmax::ssh::Node Struct Reference	1307
0.15.336.1 Detailed Description	1307
0.15.336.2 Field Documentation	1307
0.15.337 delta::Node Struct Reference	1308
0.15.337.1 Detailed Description	1308
0.15.337.2 Constructor & Destructor Documentation	1309
0.15.337.3 Member Function Documentation	1310
0.15.337.4 Field Documentation	1311
0.15.338 delta::NodePrinter< pretty > Struct Template Reference	1311
0.15.338.1 Constructor & Destructor Documentation	1311
0.15.338.2 Member Function Documentation	1311
0.15.338.3 Field Documentation	1312
0.15.339 smrat::mcsat::onecellcad::recursive::NoHeuristic Struct Reference	1312
0.15.339.1 Field Documentation	1312
0.15.340 smrat::NRAILModule< Settings > Class Template Reference	1312
0.15.340.1 Member Typedef Documentation	1317
0.15.340.2 Member Enumeration Documentation	1317
0.15.340.3 Constructor & Destructor Documentation	1318
0.15.340.4 Member Function Documentation	1318
0.15.340.5 Field Documentation	1337
0.15.341 smrat::parser::NumeralParser Struct Reference	1339
0.15.341.1 Detailed Description	1339

0.15.342 smtrat::ira::Numeric Class Reference	1339
0.15.342.1 Constructor & Destructor Documentation	1340
0.15.342.2 Member Function Documentation	1342
0.15.343 smtrat::execution::Objective Struct Reference	1346
0.15.343.1 Field Documentation	1346
0.15.344 Minisat::OccLists< Idx, Vec, Deleted > Class Template Reference	1346
0.15.344.1 Constructor & Destructor Documentation	1346
0.15.344.2 Member Function Documentation	1346
0.15.345 smtrat::mcsat::onecellcad::OneCellCAD Class Reference	1347
0.15.345.1 Constructor & Destructor Documentation	1348
0.15.345.2 Member Function Documentation	1348
0.15.345.3 Field Documentation	1349
0.15.346 benchmax::settings::OperationSettings Struct Reference	1349
0.15.346.1 Detailed Description	1349
0.15.346.2 Field Documentation	1349
0.15.347 smtrat::Optimization< Solver > Class Template Reference	1350
0.15.347.1 Constructor & Destructor Documentation	1350
0.15.347.2 Member Function Documentation	1350
0.15.348 Minisat::Option Class Reference	1351
0.15.348.1 Constructor & Destructor Documentation	1351
0.15.348.2 Member Function Documentation	1351
0.15.348.3 Friends And Related Function Documentation	1352
0.15.348.4 Field Documentation	1352
0.15.349 Minisat::Option::OptionLt Struct Reference	1352
0.15.349.1 Member Function Documentation	1352
0.15.350 smtrat::datastructures::OrderPair< T1, T2, reversed > Struct Template Reference	1353
0.15.350.1 Constructor & Destructor Documentation	1353
0.15.350.2 Member Function Documentation	1353
0.15.351 smtrat::datastructures::OrderPair< T1, T2, true > Struct Template Reference	1353
0.15.351.1 Constructor & Destructor Documentation	1353
0.15.351.2 Member Function Documentation	1353
0.15.352 smtrat::cad::Origin Class Reference	1354
0.15.352.1 Detailed Description	1354
0.15.352.2 Constructor & Destructor Documentation	1354
0.15.352.3 Member Function Documentation	1355
0.15.352.4 Friends And Related Function Documentation	1356
0.15.353 smtrat::cad::preprocessor::Origins Struct Reference	1356
0.15.353.1 Member Function Documentation	1356
0.15.353.2 Field Documentation	1357
0.15.354 smtrat::OrPathShortener Struct Reference	1357
0.15.354.1 Detailed Description	1357
0.15.354.2 Constructor & Destructor Documentation	1357

0.15.354.3 Member Function Documentation	1357
0.15.355 Minisat::OutOfMemoryException Class Reference	1358
0.15.356 smrat::parser::OutputWrapper Class Reference	1358
0.15.356.1 Constructor & Destructor Documentation	1358
0.15.356.2 Member Function Documentation	1358
0.15.357 Minisat::Map< K, D, H, E >::Pair Struct Reference	1358
0.15.357.1 Field Documentation	1359
0.15.358 smrat::pairhash< F, S, FirstHasher, SecondHasher > Struct Template Reference	1359
0.15.358.1 Detailed Description	1359
0.15.358.2 Member Function Documentation	1359
0.15.359 smrat::mcsat::ParallelExplanation< Backends > Struct Template Reference	1359
0.15.359.1 Member Function Documentation	1359
0.15.360 delta::Parser Class Reference	1360
0.15.360.1 Detailed Description	1360
0.15.360.2 Constructor & Destructor Documentation	1360
0.15.360.3 Member Function Documentation	1360
0.15.361 smrat::parser::ParserSettings Struct Reference	1361
0.15.361.1 Field Documentation	1361
0.15.362 smrat::parser::ParserState Struct Reference	1361
0.15.362.1 Constructor & Destructor Documentation	1362
0.15.362.2 Member Function Documentation	1362
0.15.362.3 Field Documentation	1363
0.15.363 smrat::PBGaussModule< Settings > Class Template Reference	1364
0.15.363.1 Member Typedef Documentation	1370
0.15.363.2 Member Enumeration Documentation	1370
0.15.363.3 Constructor & Destructor Documentation	1370
0.15.363.4 Member Function Documentation	1370
0.15.363.5 Field Documentation	1389
0.15.364 smrat::PBPPModule< Settings > Class Template Reference	1391
0.15.364.1 Member Typedef Documentation	1397
0.15.364.2 Member Enumeration Documentation	1397
0.15.364.3 Constructor & Destructor Documentation	1397
0.15.364.4 Member Function Documentation	1397
0.15.364.5 Field Documentation	1415
0.15.365 smrat::PersistentUnionFind Struct Reference	1418
0.15.365.1 Member Typedef Documentation	1418
0.15.365.2 Member Function Documentation	1419
0.15.365.3 Field Documentation	1419
0.15.366 smrat::PFEModule< Settings > Class Template Reference	1420
0.15.366.1 Member Typedef Documentation	1425
0.15.366.2 Member Enumeration Documentation	1425
0.15.366.3 Constructor & Destructor Documentation	1425

0.15.366.4 Member Function Documentation	1425
0.15.366.5 Field Documentation	1444
0.15.367 smrat::PModule Class Reference	1446
0.15.367.1 Member Enumeration Documentation	1451
0.15.367.2 Constructor & Destructor Documentation	1451
0.15.367.3 Member Function Documentation	1451
0.15.367.4 Field Documentation	1470
0.15.368 smrat::cadcells::operators::properties::poly_additional_root_outside Struct Reference	1472
0.15.368.1 Member Function Documentation	1473
0.15.368.2 Field Documentation	1473
0.15.369 smrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv Struct Reference	1473
0.15.369.1 Member Function Documentation	1473
0.15.369.2 Field Documentation	1474
0.15.370 smrat::cadcells::operators::properties::poly_irreducible_sgn_inv Struct Reference	1474
0.15.370.1 Member Function Documentation	1474
0.15.370.2 Field Documentation	1474
0.15.371 smrat::cadcells::operators::properties::poly_ord_inv Struct Reference	1474
0.15.371.1 Member Function Documentation	1475
0.15.371.2 Field Documentation	1475
0.15.372 smrat::cadcells::operators::properties::poly_ord_inv_base Struct Reference	1475
0.15.372.1 Member Function Documentation	1475
0.15.372.2 Field Documentation	1476
0.15.373 smrat::cadcells::operators::properties::poly_pdel Struct Reference	1476
0.15.373.1 Member Function Documentation	1476
0.15.373.2 Field Documentation	1476
0.15.374 smrat::cadcells::operators::properties::poly_semi_sgn_inv Struct Reference	1476
0.15.374.1 Member Function Documentation	1477
0.15.374.2 Field Documentation	1477
0.15.375 smrat::cadcells::operators::properties::poly_sgn_inv Struct Reference	1477
0.15.375.1 Member Function Documentation	1477
0.15.375.2 Field Documentation	1478
0.15.376 smrat::cadcells::representation::util::PolyDelineation Struct Reference	1478
0.15.376.1 Field Documentation	1478
0.15.377 smrat::cadcells::representation::util::PolyDelineations Struct Reference	1478
0.15.377.1 Member Function Documentation	1478
0.15.377.2 Field Documentation	1479
0.15.378 smrat::cad::ProjectionPolynomialInformation::PolyInfo Struct Reference	1479
0.15.378.1 Field Documentation	1479
0.15.379 smrat::cad::PolynomialComparator< PolynomialGetter > Struct Template Reference	1479
0.15.379.1 Constructor & Destructor Documentation	1479
0.15.379.2 Member Function Documentation	1479
0.15.380 smrat::cad::PolynomialLiftingQueue< PolynomialGetter > Class Template Reference	1479

0.15.380.1 Constructor & Destructor Documentation	1480
0.15.380.2 Member Function Documentation	1480
0.15.380.3 Friends And Related Function Documentation	1481
0.15.381 smtrat::cadcells::datastructures::PolyPool Class Reference	1481
0.15.381.1 Detailed Description	1481
0.15.381.2 Constructor & Destructor Documentation	1481
0.15.381.3 Member Function Documentation	1481
0.15.381.4 Friends And Related Function Documentation	1482
0.15.382 smtrat::cadcells::datastructures::detail::PolyProperties Struct Reference	1482
0.15.382.1 Field Documentation	1482
0.15.383 smtrat::cadcells::datastructures::PolyRef Struct Reference	1483
0.15.383.1 Detailed Description	1483
0.15.383.2 Field Documentation	1483
0.15.384 smtrat::PolyTree Class Reference	1483
0.15.384.1 Member Enumeration Documentation	1484
0.15.384.2 Constructor & Destructor Documentation	1484
0.15.384.3 Member Function Documentation	1484
0.15.385 smtrat::PolyTreeContent Class Reference	1484
0.15.385.1 Constructor & Destructor Documentation	1485
0.15.385.2 Member Function Documentation	1485
0.15.385.3 Friends And Related Function Documentation	1485
0.15.385.4 Field Documentation	1485
0.15.386 smtrat::PolyTreePool Class Reference	1485
0.15.386.1 Constructor & Destructor Documentation	1486
0.15.386.2 Member Function Documentation	1486
0.15.387 smtrat::cad::Preprocessor Class Reference	1486
0.15.387.1 Constructor & Destructor Documentation	1486
0.15.387.2 Member Function Documentation	1486
0.15.387.3 Friends And Related Function Documentation	1487
0.15.388 smtrat::cad::PreprocessorSettings Struct Reference	1487
0.15.388.1 Member Function Documentation	1488
0.15.388.2 Field Documentation	1488
0.15.389 benchmax::settings::PresetSettings Struct Reference	1488
0.15.389.1 Field Documentation	1488
0.15.390 smtrat::PriorityQueue< T, Compare > Class Template Reference	1488
0.15.390.1 Constructor & Destructor Documentation	1489
0.15.390.2 Member Function Documentation	1489
0.15.390.3 Field Documentation	1490
0.15.391 delta::Producer Class Reference	1490
0.15.391.1 Detailed Description	1491
0.15.391.2 Constructor & Destructor Documentation	1491
0.15.391.3 Member Function Documentation	1491

0.15.392 delta::ProgressBar Class Reference	1491
0.15.392.1 Detailed Description	1491
0.15.392.2 Member Function Documentation	1492
0.15.393 smrat::cad::Projection< incrementality, backtracking, Settings > Class Template Reference	1492
0.15.393.1 Member Typedef Documentation	1494
0.15.393.2 Member Function Documentation	1494
0.15.393.3 Field Documentation	1497
0.15.394 smrat::qe::cad::Projection< Settings > Class Template Reference	1497
0.15.394.1 Constructor & Destructor Documentation	1499
0.15.394.2 Member Function Documentation	1499
0.15.394.3 Friends And Related Function Documentation	1503
0.15.394.4 Field Documentation	1503
0.15.395 smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings > Class Template Reference	1503
0.15.395.1 Constructor & Destructor Documentation	1505
0.15.395.2 Member Function Documentation	1505
0.15.395.3 Friends And Related Function Documentation	1508
0.15.395.4 Field Documentation	1509
0.15.396 smrat::cad::Projection< Incrementality::FULL, BT, Settings > Class Template Reference	1509
0.15.396.1 Constructor & Destructor Documentation	1510
0.15.396.2 Member Function Documentation	1510
0.15.396.3 Friends And Related Function Documentation	1514
0.15.396.4 Field Documentation	1514
0.15.397 smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings > Class Template Reference	1514
0.15.397.1 Detailed Description	1516
0.15.397.2 Constructor & Destructor Documentation	1516
0.15.397.3 Member Function Documentation	1516
0.15.397.4 Friends And Related Function Documentation	1520
0.15.397.5 Field Documentation	1520
0.15.398 smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings > Class Template Reference	1520
0.15.398.1 Detailed Description	1521
0.15.398.2 Constructor & Destructor Documentation	1522
0.15.398.3 Member Function Documentation	1522
0.15.398.4 Friends And Related Function Documentation	1525
0.15.398.5 Field Documentation	1525
0.15.399 smrat::cad::Projection< Incrementality::SIMPLE, BT, Settings > Class Template Reference	1526
0.15.399.1 Constructor & Destructor Documentation	1527
0.15.399.2 Member Function Documentation	1527
0.15.399.3 Friends And Related Function Documentation	1531
0.15.399.4 Field Documentation	1531

0.15.400 smrat::cad::projection_compare::ProjectionComparator< Strategy > Struct Template Reference	1531
0.15.401 smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D > Struct Reference	1531
0.15.401.1 Member Function Documentation	1532
0.15.402 smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD > Struct Reference	1532
0.15.402.1 Member Function Documentation	1532
0.15.403 smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID > Struct Reference	1532
0.15.403.1 Member Function Documentation	1532
0.15.404 smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD > Struct Reference	1532
0.15.404.1 Member Function Documentation	1532
0.15.405 smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD > Struct Reference	1533
0.15.405.1 Member Function Documentation	1533
0.15.406 smrat::cad::projection_compare::ProjectionComparator_Impl< Args > Struct Template Reference	1533
0.15.406.1 Member Function Documentation	1533
0.15.407 smrat::cad::ProjectionGlobalInformation Class Reference	1533
0.15.407.1 Member Typedef Documentation	1534
0.15.407.2 Member Function Documentation	1534
0.15.407.3 Field Documentation	1534
0.15.408 smrat::cad::ProjectionInformation Class Reference	1535
0.15.408.1 Member Function Documentation	1535
0.15.409 smrat::cad::ProjectionLevelInformation Class Reference	1537
0.15.409.1 Member Function Documentation	1537
0.15.410 smrat::cad::ProjectionOperator Struct Reference	1537
0.15.410.1 Member Function Documentation	1537
0.15.411 smrat::cad::ProjectionPolynomialInformation Class Reference	1538
0.15.411.1 Member Function Documentation	1538
0.15.412 smrat::cadcells::datastructures::Projections Class Reference	1539
0.15.412.1 Detailed Description	1539
0.15.412.2 Constructor & Destructor Documentation	1539
0.15.412.3 Member Function Documentation	1540
0.15.413 smrat::cadcells::operators::mccallum_filtered_Impl::PropertiesSet Struct Reference	1542
0.15.413.1 Member Typedef Documentation	1542
0.15.414 smrat::cadcells::operators::PropertiesSet< Op > Struct Template Reference	1542
0.15.415 smrat::cadcells::operators::PropertiesSet< op::mccallum > Struct Reference	1542
0.15.415.1 Member Typedef Documentation	1542
0.15.416 smrat::cadcells::datastructures::PropertiesT< Ts > Struct Template Reference	1542
0.15.416.1 Detailed Description	1542
0.15.417 smrat::cadcells::datastructures::PropertiesT< T, Ts... > Struct Template Reference	1543

0.15.417.1 Field Documentation	1543
0.15.418 smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag > Struct Template Reference	1543
0.15.419 smtrat::cadcells::datastructures::PropertiesTContent< T, false > Struct Template Reference	1543
0.15.419.1 Member Typedef Documentation	1543
0.15.420 smtrat::cadcells::datastructures::PropertiesTContent< T, true > Struct Template Reference	1543
0.15.420.1 Member Typedef Documentation	1543
0.15.421 smtrat::cadcells::datastructures::property_hash< T > Struct Template Reference	1544
0.15.421.1 Member Function Documentation	1544
0.15.422 smtrat::PseudoBoolEncoder Class Reference	1544
0.15.422.1 Detailed Description	1544
0.15.422.2 Member Function Documentation	1544
0.15.422.3 Field Documentation	1545
0.15.423 smtrat::PseudoBoolNormalizer Class Reference	1545
0.15.423.1 Member Function Documentation	1545
0.15.424 smtrat::parser::QEParser Struct Reference	1546
0.15.424.1 Constructor & Destructor Documentation	1546
0.15.424.2 Member Function Documentation	1546
0.15.424.3 Field Documentation	1546
0.15.425 smtrat::parser::QualifiedIdentifierParser Struct Reference	1547
0.15.425.1 Constructor & Destructor Documentation	1547
0.15.425.2 Member Function Documentation	1547
0.15.425.3 Field Documentation	1547
0.15.426 smtrat::expression::QuantifierExpression Struct Reference	1547
0.15.426.1 Constructor & Destructor Documentation	1548
0.15.426.2 Field Documentation	1548
0.15.427 smtrat::parser::QuantifierParser Struct Reference	1548
0.15.427.1 Constructor & Destructor Documentation	1548
0.15.428 Minisat::Queue< T > Class Template Reference	1548
0.15.428.1 Member Typedef Documentation	1549
0.15.428.2 Constructor & Destructor Documentation	1549
0.15.428.3 Member Function Documentation	1549
0.15.429 smtrat::mcsat::icp::QueueEntry Struct Reference	1549
0.15.429.1 Field Documentation	1550
0.15.430 benchmax::RandomizationAdaptor< T > Class Template Reference	1550
0.15.430.1 Detailed Description	1550
0.15.430.2 Constructor & Destructor Documentation	1550
0.15.430.3 Member Function Documentation	1550
0.15.431 smtrat::RationalCapsule Class Reference	1551
0.15.431.1 Constructor & Destructor Documentation	1551
0.15.431.2 Member Function Documentation	1551
0.15.431.3 Field Documentation	1551

0.15.432 smrat::parser::RationalPolicies Struct Reference	1552
0.15.432.1 Detailed Description	1552
0.15.432.2 Member Function Documentation	1552
0.15.433 smrat::mcsat::oncelcad::RealAlgebraicPoint< Number > Class Template Reference	1552
0.15.433.1 Detailed Description	1553
0.15.433.2 Constructor & Destructor Documentation	1553
0.15.433.3 Member Function Documentation	1553
0.15.434 smrat::fixedsize_allocator< T >::rebind< U > Struct Template Reference	1554
0.15.434.1 Member Typedef Documentation	1554
0.15.435 smrat::mcsat::oncelcad::recursive::RecursiveCAD Class Reference	1554
0.15.435.1 Member Function Documentation	1555
0.15.435.2 Field Documentation	1558
0.15.436 smrat::cad::projection::Reducta< Poly > Struct Template Reference	1558
0.15.436.1 Detailed Description	1559
0.15.436.2 Constructor & Destructor Documentation	1559
0.15.436.3 Field Documentation	1559
0.15.437 Minisat::RegionAllocator< T > Class Template Reference	1559
0.15.437.1 Member Typedef Documentation	1559
0.15.437.2 Member Enumeration Documentation	1560
0.15.437.3 Constructor & Destructor Documentation	1560
0.15.437.4 Member Function Documentation	1560
0.15.438 smrat::NNFRewriter::remove_xor_first_arg Struct Reference	1561
0.15.438.1 Constructor & Destructor Documentation	1561
0.15.439 smrat::ReplaceVariablesRewriter Struct Reference	1561
0.15.439.1 Member Typedef Documentation	1562
0.15.439.2 Constructor & Destructor Documentation	1562
0.15.439.3 Member Function Documentation	1562
0.15.440 delta::ErrorHandler::result< typename > Struct Template Reference	1563
0.15.440.1 Member Typedef Documentation	1563
0.15.441 smrat::parser::ErrorHandler::result< typename > Struct Template Reference	1563
0.15.441.1 Member Typedef Documentation	1563
0.15.442 smrat::cad::preprocessor::ResultantRule Class Reference	1563
0.15.442.1 Constructor & Destructor Documentation	1563
0.15.442.2 Member Function Documentation	1564
0.15.443 benchmax::Results Class Reference	1564
0.15.443.1 Detailed Description	1564
0.15.443.2 Member Function Documentation	1564
0.15.444 smrat::RNSEncoder Class Reference	1565
0.15.444.1 Constructor & Destructor Documentation	1565
0.15.444.2 Member Function Documentation	1565
0.15.444.3 Field Documentation	1566
0.15.445 smrat::cadcells::operators::properties::root_ordering_holds Struct Reference	1566

0.15.445.1 Member Function Documentation	1566
0.15.445.2 Field Documentation	1567
0.15.446 smtrat::cadcells::operators::properties::root_well_def Struct Reference	1567
0.15.446.1 Member Function Documentation	1567
0.15.446.2 Field Documentation	1567
0.15.447 smtrat::cadcells::datastructures::RootFunction Class Reference	1568
0.15.447.1 Constructor & Destructor Documentation	1568
0.15.447.2 Member Function Documentation	1568
0.15.447.3 Friends And Related Function Documentation	1569
0.15.448 smtrat::mcsat::arithmetic::RootIndexer< RANT > Class Template Reference	1570
0.15.448.1 Member Function Documentation	1570
0.15.449 smtrat::cad::Sample Class Reference	1570
0.15.449.1 Constructor & Destructor Documentation	1571
0.15.449.2 Member Function Documentation	1571
0.15.449.3 Friends And Related Function Documentation	1573
0.15.450 smtrat::cad::sample_compare::SampleComparator< Iterator, Strategy > Struct Template Reference	1573
0.15.451 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::LS > > Struct Template Reference	1573
0.15.451.1 Member Function Documentation	1573
0.15.452 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::LT > > Struct Template Reference	1573
0.15.452.1 Member Function Documentation	1573
0.15.453 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::LTA > > Struct Template Reference	1573
0.15.453.1 Member Function Documentation	1574
0.15.454 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::LTS > > Struct Template Reference	1574
0.15.454.1 Member Function Documentation	1574
0.15.455 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::LTSA > > Struct Template Reference	1574
0.15.455.1 Member Function Documentation	1574
0.15.456 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::S > > Struct Template Reference	1574
0.15.456.1 Member Function Documentation	1574
0.15.457 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::T > > Struct Template Reference	1575
0.15.457.1 Member Function Documentation	1575
0.15.458 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::TLSA > > Struct Template Reference	1575
0.15.458.1 Member Function Documentation	1575
0.15.459 smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::TS > > Struct Template Reference	1575
0.15.459.1 Member Function Documentation	1575

0.15.460 smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::TSA > Struct Template Reference	1576
0.15.460.1 Member Function Documentation	1576
0.15.461 smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::Type > Struct Template Reference	1576
0.15.461.1 Member Function Documentation	1576
0.15.462 smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy< ::Value > Struct Template Reference	1576
0.15.462.1 Member Function Documentation	1577
0.15.463 smrat::cad::sample_compare::SampleComparator_Impl< It, Args > Struct Template Reference	1577
0.15.463.1 Member Function Documentation	1577
0.15.464 smrat::cadcells::datastructures::SampledDerivation< Properties > Class Template Reference	1577
0.15.464.1 Detailed Description	1578
0.15.464.2 Constructor & Destructor Documentation	1578
0.15.464.3 Member Function Documentation	1578
0.15.465 smrat::SampledDerivationRefCompare Struct Reference	1580
0.15.465.1 Member Function Documentation	1580
0.15.466 smrat::cad::SampleIteratorQueue< Iterator, Comparator > Class Template Reference	1580
0.15.466.1 Member Function Documentation	1581
0.15.467 smrat::sampling< S > Struct Template Reference	1582
0.15.467.1 Member Function Documentation	1582
0.15.468 smrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING > Struct Reference	1582
0.15.468.1 Member Function Documentation	1582
0.15.469 smrat::SATModule< Settings > Class Template Reference	1582
0.15.469.1 Detailed Description	1588
0.15.469.2 Member Typedef Documentation	1588
0.15.469.3 Member Enumeration Documentation	1589
0.15.469.4 Constructor & Destructor Documentation	1589
0.15.469.5 Member Function Documentation	1589
0.15.469.6 Friends And Related Function Documentation	1612
0.15.469.7 Field Documentation	1612
0.15.470 smrat::parser::ScriptParser< Callee > Struct Template Reference	1614
0.15.470.1 Constructor & Destructor Documentation	1614
0.15.470.2 Member Function Documentation	1615
0.15.470.3 Field Documentation	1615
0.15.471 smrat::parser::ParserState::ScriptScope Struct Reference	1616
0.15.471.1 Constructor & Destructor Documentation	1616
0.15.471.2 Member Function Documentation	1617
0.15.472 smrat::mcsat::onecellcad::Section Struct Reference	1617
0.15.472.1 Detailed Description	1617
0.15.472.2 Field Documentation	1617

0.15.473 smrat::onecellcad::recursive::Section Struct Reference	1617
0.15.473.1 Detailed Description	1618
0.15.473.2 Field Documentation	1618
0.15.474 smrat::mcsat::onecellcad::levelwise::SectionHeuristic1 Struct Reference	1618
0.15.474.1 Field Documentation	1618
0.15.475 smrat::mcsat::onecellcad::levelwise::SectionHeuristic2 Struct Reference	1618
0.15.475.1 Field Documentation	1618
0.15.476 smrat::mcsat::onecellcad::levelwise::SectionHeuristic3 Struct Reference	1618
0.15.476.1 Field Documentation	1619
0.15.477 smrat::mcsat::onecellcad::Sector Struct Reference	1619
0.15.477.1 Detailed Description	1619
0.15.477.2 Field Documentation	1619
0.15.478 smrat::onecellcad::recursive::Sector Struct Reference	1619
0.15.478.1 Detailed Description	1620
0.15.478.2 Member Function Documentation	1620
0.15.478.3 Field Documentation	1620
0.15.479 smrat::mcsat::onecellcad::levelwise::SectorHeuristic1 Struct Reference	1620
0.15.479.1 Field Documentation	1620
0.15.480 smrat::mcsat::onecellcad::levelwise::SectorHeuristic2 Struct Reference	1620
0.15.480.1 Field Documentation	1620
0.15.481 smrat::mcsat::onecellcad::levelwise::SectorHeuristic3 Struct Reference	1621
0.15.481.1 Field Documentation	1621
0.15.482 smrat::subtropical::Separator Struct Reference	1621
0.15.482.1 Detailed Description	1621
0.15.482.2 Constructor & Destructor Documentation	1621
0.15.482.3 Field Documentation	1621
0.15.483 smrat::mcsat::SequentialAssignment< Backends > Struct Template Reference	1622
0.15.484 smrat::mcsat::SequentialExplanation< Backends > Struct Template Reference	1622
0.15.485 benchmax::settings::Settings Struct Reference	1622
0.15.485.1 Detailed Description	1622
0.15.486 delta::Settings Class Reference	1622
0.15.486.1 Detailed Description	1622
0.15.486.2 Constructor & Destructor Documentation	1622
0.15.486.3 Member Function Documentation	1624
0.15.486.4 Friends And Related Function Documentation	1626
0.15.487 smrat::settings::Settings Struct Reference	1626
0.15.488 smrat::SettingsComponents Class Reference	1626
0.15.488.1 Member Function Documentation	1626
0.15.489 benchmax::SettingsParser Class Reference	1627
0.15.489.1 Detailed Description	1627
0.15.490 smrat::SettingsParser Class Reference	1627
0.15.491 smrat::CNFerModule::SettingsType Struct Reference	1627

0.15.491.1 Field Documentation	1627
0.15.492 smrat::parser::SExpressionParser Struct Reference	1627
0.15.492.1 Constructor & Destructor Documentation	1627
0.15.492.2 Field Documentation	1628
0.15.493 smrat::parser::SExpressionSequence< T > Struct Template Reference	1628
0.15.493.1 Constructor & Destructor Documentation	1628
0.15.493.2 Field Documentation	1628
0.15.494 smrat::ShortFormulaEncoder Class Reference	1628
0.15.494.1 Constructor & Destructor Documentation	1629
0.15.494.2 Member Function Documentation	1629
0.15.494.3 Field Documentation	1630
0.15.495 benchmax::simple_parser Class Reference	1630
0.15.495.1 Constructor & Destructor Documentation	1630
0.15.495.2 Member Function Documentation	1630
0.15.496 smrat::parser::Theories::SimpleSortAdder Struct Reference	1631
0.15.496.1 Detailed Description	1631
0.15.496.2 Constructor & Destructor Documentation	1631
0.15.496.3 Member Function Documentation	1631
0.15.496.4 Field Documentation	1631
0.15.497 smrat::parser::SimpleSymbolParser Struct Reference	1631
0.15.497.1 Detailed Description	1632
0.15.497.2 Constructor & Destructor Documentation	1632
0.15.497.3 Field Documentation	1632
0.15.498 smrat::expression::simplifier::Simplifier Class Reference	1632
0.15.498.1 Member Function Documentation	1632
0.15.499 smrat::expression::simplifier::SimplifierChainCaller< chainID > Struct Template Reference	1632
0.15.499.1 Member Function Documentation	1632
0.15.499.2 Field Documentation	1633
0.15.500 smrat::expression::simplifier::SimplifierChainCaller< 0 > Struct Reference	1633
0.15.500.1 Member Function Documentation	1633
0.15.501 smrat::expression::simplifier::SingletonSimplifier Struct Reference	1633
0.15.501.1 Member Function Documentation	1634
0.15.502 smrat::cad::sample_compare::size Struct Reference	1635
0.15.503 delta::Skipper Struct Reference	1635
0.15.503.1 Detailed Description	1635
0.15.503.2 Constructor & Destructor Documentation	1635
0.15.503.3 Field Documentation	1635
0.15.504 smrat::parser::Skipper Struct Reference	1635
0.15.504.1 Constructor & Destructor Documentation	1636
0.15.504.2 Field Documentation	1636
0.15.505 benchmax::SlurmBackend Class Reference	1636
0.15.505.1 Detailed Description	1636

0.15.505.2 Member Function Documentation	1637
0.15.505.3 Field Documentation	1637
0.15.506 benchmax::settings::SlurmBackendSettings Struct Reference	1638
0.15.506.1 Detailed Description	1638
0.15.506.2 Field Documentation	1638
0.15.507 smtrat::parser::SMTLIBParser Class Reference	1639
0.15.507.1 Constructor & Destructor Documentation	1640
0.15.507.2 Member Function Documentation	1640
0.15.507.3 Field Documentation	1642
0.15.508 benchmax::SMTRAT Class Reference	1642
0.15.508.1 Detailed Description	1643
0.15.508.2 Constructor & Destructor Documentation	1643
0.15.508.3 Member Function Documentation	1643
0.15.508.4 Field Documentation	1645
0.15.509 benchmax::SMTRAT_Analyzer Class Reference	1645
0.15.509.1 Detailed Description	1646
0.15.509.2 Constructor & Destructor Documentation	1646
0.15.509.3 Member Function Documentation	1646
0.15.509.4 Field Documentation	1647
0.15.510 benchmax::SMTRAT_OPB Class Reference	1647
0.15.510.1 Detailed Description	1648
0.15.510.2 Constructor & Destructor Documentation	1648
0.15.510.3 Member Function Documentation	1648
0.15.510.4 Field Documentation	1650
0.15.511 smtrat::execution::SoftAssertion Struct Reference	1650
0.15.511.1 Field Documentation	1650
0.15.512 smtrat::settings::SolverSettings Struct Reference	1650
0.15.512.1 Field Documentation	1651
0.15.513 smtrat::parser::SortedVariableParser Struct Reference	1651
0.15.513.1 Constructor & Destructor Documentation	1651
0.15.513.2 Field Documentation	1651
0.15.514 smtrat::parser::SortParser Struct Reference	1652
0.15.514.1 Constructor & Destructor Documentation	1652
0.15.514.2 Member Function Documentation	1652
0.15.515 smtrat::parser::SpecConstantParser Struct Reference	1652
0.15.515.1 Constructor & Destructor Documentation	1652
0.15.515.2 Field Documentation	1653
0.15.516 smtrat::SplitSOSModule< Settings > Class Template Reference	1653
0.15.516.1 Member Typedef Documentation	1658
0.15.516.2 Member Enumeration Documentation	1658
0.15.516.3 Constructor & Destructor Documentation	1658
0.15.516.4 Member Function Documentation	1659

0.15.516.5 Field Documentation	1677
0.15.517 benchmax::SSHBackend Class Reference	1680
0.15.517.1 Constructor & Destructor Documentation	1680
0.15.517.2 Member Function Documentation	1681
0.15.517.3 Field Documentation	1682
0.15.518 benchmax::settings::SSHBackendSettings Struct Reference	1682
0.15.518.1 Detailed Description	1682
0.15.518.2 Field Documentation	1682
0.15.519 benchmax::ssh::SSHConnection Class Reference	1683
0.15.519.1 Detailed Description	1683
0.15.519.2 Constructor & Destructor Documentation	1683
0.15.519.3 Member Function Documentation	1684
0.15.520 smtrat::vs::State Class Reference	1685
0.15.520.1 Member Typedef Documentation	1689
0.15.520.2 Member Enumeration Documentation	1689
0.15.520.3 Constructor & Destructor Documentation	1689
0.15.520.4 Member Function Documentation	1690
0.15.521 smtrat::StaticUnionFind Struct Reference	1710
0.15.521.1 Member Typedef Documentation	1710
0.15.521.2 Member Function Documentation	1710
0.15.522 smtrat::statistics::StatisticsSettings Struct Reference	1711
0.15.522.1 Field Documentation	1711
0.15.523 smtrat::StrategyGraph Class Reference	1711
0.15.523.1 Constructor & Destructor Documentation	1711
0.15.523.2 Member Function Documentation	1711
0.15.523.3 Friends And Related Function Documentation	1712
0.15.524 delta::String Class Reference	1712
0.15.524.1 Detailed Description	1712
0.15.524.2 Member Function Documentation	1713
0.15.525 Minisat::StringOption Class Reference	1713
0.15.525.1 Constructor & Destructor Documentation	1713
0.15.525.2 Member Function Documentation	1714
0.15.525.3 Field Documentation	1714
0.15.526 smtrat::parser::StringParser Struct Reference	1714
0.15.526.1 Detailed Description	1715
0.15.526.2 Constructor & Destructor Documentation	1715
0.15.526.3 Field Documentation	1715
0.15.527 smtrat::STropModule< Settings > Class Template Reference	1715
0.15.527.1 Member Typedef Documentation	1720
0.15.527.2 Member Enumeration Documentation	1720
0.15.527.3 Constructor & Destructor Documentation	1720
0.15.527.4 Member Function Documentation	1720

0.15.527.5 Field Documentation	1740
0.15.528 benchmax::slurm::SubmitfileProperties Struct Reference	1742
0.15.528.1 Detailed Description	1742
0.15.528.2 Field Documentation	1742
0.15.529 smtrat::vs::Substitution Class Reference	1743
0.15.529.1 Member Enumeration Documentation	1744
0.15.529.2 Constructor & Destructor Documentation	1744
0.15.529.3 Member Function Documentation	1745
0.15.530 smtrat::vs::substitutionPointerEqual Struct Reference	1747
0.15.530.1 Member Function Documentation	1747
0.15.531 smtrat::vs::substitutionPointerHash Struct Reference	1747
0.15.531.1 Member Function Documentation	1747
0.15.532 smtrat::cadcells::datastructures::SymbolicInterval Class Reference	1748
0.15.532.1 Detailed Description	1748
0.15.532.2 Constructor & Destructor Documentation	1748
0.15.532.3 Member Function Documentation	1748
0.15.533 delta::SymbolParser Struct Reference	1749
0.15.533.1 Detailed Description	1749
0.15.533.2 Constructor & Destructor Documentation	1749
0.15.533.3 Field Documentation	1750
0.15.534 smtrat::parser::SymbolParser Struct Reference	1750
0.15.534.1 Constructor & Destructor Documentation	1750
0.15.534.2 Field Documentation	1750
0.15.535 smtrat::SymmetryModule< Settings > Class Template Reference	1750
0.15.535.1 Member Typedef Documentation	1755
0.15.535.2 Member Enumeration Documentation	1755
0.15.535.3 Constructor & Destructor Documentation	1756
0.15.535.4 Member Function Documentation	1756
0.15.535.5 Field Documentation	1776
0.15.536 smtrat::lra::Tableau< Settings, T1, T2 > Class Template Reference	1778
0.15.536.1 Constructor & Destructor Documentation	1780
0.15.536.2 Member Function Documentation	1780
0.15.537 smtrat::lra::TableauEntry< T1, T2 > Class Template Reference	1796
0.15.537.1 Constructor & Destructor Documentation	1796
0.15.537.2 Member Function Documentation	1797
0.15.538 smtrat::cadcells::datastructures::TaggedIndexedRoot Struct Reference	1799
0.15.538.1 Field Documentation	1799
0.15.539 smtrat::mcsat::onecellcad::TagPoly Struct Reference	1799
0.15.539.1 Detailed Description	1800
0.15.539.2 Field Documentation	1800
0.15.540 smtrat::Task Class Reference	1800
0.15.540.1 Constructor & Destructor Documentation	1800

0.15.540.2 Member Function Documentation	1800
0.15.541 delta::TempFilenameGenerator Class Reference	1801
0.15.541.1 Detailed Description	1801
0.15.541.2 Constructor & Destructor Documentation	1801
0.15.541.3 Member Function Documentation	1801
0.15.542 smrat::parser::TermParser Struct Reference	1802
0.15.542.1 Member Typedef Documentation	1802
0.15.542.2 Constructor & Destructor Documentation	1802
0.15.542.3 Field Documentation	1802
0.15.543 smrat::mcsat::vs::helper::TestCandidate Struct Reference	1803
0.15.543.1 Member Function Documentation	1803
0.15.543.2 Field Documentation	1803
0.15.544 smrat::parser::Theories Struct Reference	1804
0.15.544.1 Detailed Description	1804
0.15.544.2 Member Typedef Documentation	1805
0.15.544.3 Constructor & Destructor Documentation	1805
0.15.544.4 Member Function Documentation	1805
0.15.545 smrat::parser::TheoryError Struct Reference	1807
0.15.545.1 Member Function Documentation	1807
0.15.545.2 Friends And Related Function Documentation	1807
0.15.546 smrat::mcsat::TheoryLevel Struct Reference	1807
0.15.546.1 Field Documentation	1808
0.15.547 smrat::Module::TheoryPropagation Struct Reference	1808
0.15.547.1 Constructor & Destructor Documentation	1808
0.15.547.2 Member Function Documentation	1809
0.15.547.3 Field Documentation	1809
0.15.548 smrat::TheoryVarSchedulerStatic< vot > Class Template Reference	1809
0.15.548.1 Detailed Description	1810
0.15.548.2 Constructor & Destructor Documentation	1810
0.15.548.3 Member Function Documentation	1810
0.15.548.4 Field Documentation	1811
0.15.549 smrat::ThreadPool Class Reference	1813
0.15.549.1 Constructor & Destructor Documentation	1813
0.15.549.2 Member Function Documentation	1813
0.15.550 smrat::cad::debug::TikzBasePrinter Class Reference	1813
0.15.550.1 Member Function Documentation	1814
0.15.550.2 Field Documentation	1815
0.15.551 smrat::cad::debug::TikzDAGPrinter Class Reference	1815
0.15.551.1 Member Function Documentation	1815
0.15.551.2 Field Documentation	1816
0.15.552 smrat::cad::debug::TikzHistoryPrinter Class Reference	1816
0.15.552.1 Member Function Documentation	1816

0.15.553 smtrat::cad::debug::TikzTreePrinter Class Reference	1817
0.15.553.1 Member Function Documentation	1818
0.15.553.2 Field Documentation	1818
0.15.554 benchmax::Tool Class Reference	1819
0.15.554.1 Detailed Description	1820
0.15.554.2 Constructor & Destructor Documentation	1820
0.15.554.3 Member Function Documentation	1820
0.15.554.4 Friends And Related Function Documentation	1821
0.15.554.5 Field Documentation	1822
0.15.555 benchmax::settings::ToolSettings Struct Reference	1822
0.15.555.1 Detailed Description	1822
0.15.555.2 Field Documentation	1823
0.15.556 smtrat::TotalizerEncoder Class Reference	1823
0.15.556.1 Detailed Description	1824
0.15.556.2 Constructor & Destructor Documentation	1824
0.15.556.3 Member Function Documentation	1824
0.15.556.4 Field Documentation	1825
0.15.557 smtrat::TotalizerTree Class Reference	1825
0.15.557.1 Constructor & Destructor Documentation	1825
0.15.557.2 Member Function Documentation	1825
0.15.558 smtrat::cad::projection_compare::type Struct Reference	1826
0.15.559 smtrat::cad::sample_compare::type Struct Reference	1826
0.15.560 smtrat::uf_impl::uf_data_wrapper< T, compression > Class Template Reference	1826
0.15.560.1 Detailed Description	1826
0.15.560.2 Member Typedef Documentation	1826
0.15.560.3 Constructor & Destructor Documentation	1826
0.15.560.4 Member Function Documentation	1826
0.15.561 smtrat::uf_impl::uf_data_wrapper< void, compression > Class Template Reference	1827
0.15.561.1 Detailed Description	1827
0.15.561.2 Constructor & Destructor Documentation	1827
0.15.561.3 Member Function Documentation	1828
0.15.562 smtrat::uf_impl::uf_rank_and_class< compression > Class Template Reference	1828
0.15.562.1 Detailed Description	1828
0.15.562.2 Constructor & Destructor Documentation	1829
0.15.562.3 Member Function Documentation	1829
0.15.563 smtrat::uf_impl::uf_rank_and_class< true > Class Reference	1829
0.15.563.1 Detailed Description	1829
0.15.563.2 Constructor & Destructor Documentation	1829
0.15.563.3 Member Function Documentation	1830
0.15.564 smtrat::UFCegarModule< Settings > Class Template Reference	1830
0.15.564.1 Member Typedef Documentation	1835
0.15.564.2 Member Enumeration Documentation	1835

0.15.564.3 Constructor & Destructor Documentation	1835
0.15.564.4 Member Function Documentation	1836
0.15.564.5 Field Documentation	1854
0.15.565 smrat::UFRewriter Struct Reference	1856
0.15.565.1 Member Typedef Documentation	1856
0.15.565.2 Member Function Documentation	1857
0.15.566 smrat::expression::UnaryExpression Struct Reference	1857
0.15.566.1 Constructor & Destructor Documentation	1857
0.15.566.2 Field Documentation	1857
0.15.567 smrat::cad::debug::UnifiedData Struct Reference	1857
0.15.567.1 Member Function Documentation	1858
0.15.567.2 Field Documentation	1858
0.15.568 smrat::parser::types::UninterpretedTheory Struct Reference	1858
0.15.568.1 Detailed Description	1858
0.15.568.2 Member Typedef Documentation	1858
0.15.569 smrat::parser::UninterpretedTheory Struct Reference	1859
0.15.569.1 Detailed Description	1860
0.15.569.2 Constructor & Destructor Documentation	1860
0.15.569.3 Member Function Documentation	1860
0.15.569.4 Field Documentation	1861
0.15.570 smrat::union_find< T, no_compression, MaxTracebackLength > Class Template Reference	1862
0.15.570.1 Detailed Description	1863
0.15.570.2 Constructor & Destructor Documentation	1863
0.15.570.3 Member Function Documentation	1864
0.15.571 smrat::UnionFindInterface< T, Implementation > Struct Template Reference	1866
0.15.571.1 Member Typedef Documentation	1866
0.15.571.2 Constructor & Destructor Documentation	1866
0.15.571.3 Member Function Documentation	1867
0.15.572 smrat::UnionFindModule< Settings > Class Template Reference	1867
0.15.572.1 Member Typedef Documentation	1872
0.15.572.2 Member Enumeration Documentation	1872
0.15.572.3 Constructor & Destructor Documentation	1873
0.15.572.4 Member Function Documentation	1873
0.15.572.5 Field Documentation	1892
0.15.573 smrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher Struct Reference	1894
0.15.573.1 Member Function Documentation	1894
0.15.574 smrat::UnsatCore< Solver, Strategy > Class Template Reference	1894
0.15.574.1 Constructor & Destructor Documentation	1895
0.15.574.2 Member Function Documentation	1895
0.15.575 smrat::unsatcore::UnsatCoreBackend< Solver, Strategy > Class Template Reference	1895
0.15.576 smrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion > Class Template Reference	1895

0.15.576.1 Detailed Description	1896
0.15.576.2 Constructor & Destructor Documentation	1896
0.15.576.3 Member Function Documentation	1896
0.15.577 smrat::vs::unsignedTripleCmp Struct Reference	1896
0.15.577.1 Member Function Documentation	1896
0.15.578 smrat::parser::UserFunctionInstantiator Struct Reference	1897
0.15.578.1 Constructor & Destructor Documentation	1897
0.15.578.2 Member Function Documentation	1897
0.15.578.3 Field Documentation	1898
0.15.579 smrat::validation::ValidationCollector Class Reference	1898
0.15.579.1 Member Function Documentation	1898
0.15.580 smrat::validation::ValidationPoint Class Reference	1898
0.15.580.1 Member Function Documentation	1899
0.15.581 smrat::validation::ValidationPrinter< SOF > Struct Template Reference	1900
0.15.582 smrat::validation::ValidationSettings Struct Reference	1900
0.15.582.1 Member Function Documentation	1900
0.15.582.2 Field Documentation	1900
0.15.583 smrat::lra::Value< T > Class Template Reference	1900
0.15.583.1 Constructor & Destructor Documentation	1901
0.15.583.2 Member Function Documentation	1902
0.15.584 smrat::lra::Variable< T1, T2 > Class Template Reference	1908
0.15.584.1 Constructor & Destructor Documentation	1909
0.15.584.2 Member Function Documentation	1910
0.15.585 smrat::vb::Variable< T > Class Template Reference	1918
0.15.585.1 Detailed Description	1919
0.15.585.2 Constructor & Destructor Documentation	1919
0.15.585.3 Member Function Documentation	1919
0.15.586 smrat::vb::VariableBounds< T > Class Template Reference	1921
0.15.586.1 Detailed Description	1922
0.15.586.2 Member Typedef Documentation	1922
0.15.586.3 Constructor & Destructor Documentation	1922
0.15.586.4 Member Function Documentation	1923
0.15.587 smrat::VariableCapsule Class Reference	1928
0.15.587.1 Constructor & Destructor Documentation	1928
0.15.587.2 Member Function Documentation	1928
0.15.587.3 Field Documentation	1929
0.15.588 smrat::mcsat::variableordering::VariableIDs Struct Reference	1929
0.15.588.1 Member Function Documentation	1929
0.15.588.2 Field Documentation	1929
0.15.589 smrat::VariableRewriteRule Class Reference	1929
0.15.589.1 Constructor & Destructor Documentation	1930
0.15.589.2 Field Documentation	1930

0.15.590 smrat::variant_hash_visitor Struct Reference	1930
0.15.590.1 Member Function Documentation	1930
0.15.591 smrat::parser::conversion::VariantConverter< Res > Struct Template Reference	1931
0.15.591.1 Detailed Description	1931
0.15.591.2 Member Typedef Documentation	1931
0.15.591.3 Member Function Documentation	1931
0.15.591.4 Field Documentation	1932
0.15.592 smrat::VariantMap< Key, Value > Class Template Reference	1932
0.15.592.1 Detailed Description	1932
0.15.592.2 Member Function Documentation	1933
0.15.592.3 Field Documentation	1934
0.15.593 smrat::parser::conversion::VariantVariantConverter< Res > Struct Template Reference .	1934
0.15.593.1 Detailed Description	1935
0.15.593.2 Member Typedef Documentation	1935
0.15.593.3 Member Function Documentation	1935
0.15.594 smrat::VarSchedulerBase Class Reference	1935
0.15.594.1 Detailed Description	1936
0.15.594.2 Constructor & Destructor Documentation	1936
0.15.594.3 Member Function Documentation	1936
0.15.594.4 Field Documentation	1937
0.15.595 smrat::VarSchedulerFixedRandom Class Reference	1938
0.15.595.1 Constructor & Destructor Documentation	1939
0.15.595.2 Member Function Documentation	1939
0.15.595.3 Field Documentation	1940
0.15.596 smrat::VarSchedulerMcsatActivityPreferTheory< vot > Class Template Reference . . .	1941
0.15.596.1 Detailed Description	1942
0.15.596.2 Constructor & Destructor Documentation	1942
0.15.596.3 Member Function Documentation	1942
0.15.596.4 Field Documentation	1943
0.15.597 smrat::VarSchedulerMcsatBase Class Reference	1945
0.15.597.1 Detailed Description	1946
0.15.597.2 Constructor & Destructor Documentation	1946
0.15.597.3 Member Function Documentation	1946
0.15.597.4 Field Documentation	1947
0.15.598 smrat::VarSchedulerMcsatBooleanFirst< vot > Class Template Reference	1948
0.15.598.1 Detailed Description	1949
0.15.598.2 Constructor & Destructor Documentation	1949
0.15.598.3 Member Function Documentation	1949
0.15.598.4 Field Documentation	1950
0.15.599 smrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler > Class Template Reference .	1952
0.15.599.1 Detailed Description	1952
0.15.599.2 Constructor & Destructor Documentation	1953

0.15.599.3 Member Function Documentation	1953
0.15.599.4 Field Documentation	1954
0.15.600 smrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities > Class Template Reference	1955
0.15.600.1 Detailed Description	1956
0.15.600.2 Constructor & Destructor Documentation	1956
0.15.600.3 Member Function Documentation	1956
0.15.600.4 Field Documentation	1958
0.15.601 smrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot > Class Template Reference	1959
0.15.601.1 Detailed Description	1960
0.15.601.2 Constructor & Destructor Documentation	1960
0.15.601.3 Member Function Documentation	1960
0.15.601.4 Field Documentation	1961
0.15.602 smrat::VarSchedulerMinisat Class Reference	1963
0.15.602.1 Detailed Description	1963
0.15.602.2 Constructor & Destructor Documentation	1963
0.15.602.3 Member Function Documentation	1964
0.15.602.4 Field Documentation	1965
0.15.603 smrat::VarSchedulerRandom Class Reference	1966
0.15.603.1 Detailed Description	1967
0.15.603.2 Constructor & Destructor Documentation	1967
0.15.603.3 Member Function Documentation	1967
0.15.603.4 Field Documentation	1968
0.15.604 smrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic > Class Template Reference	1969
0.15.604.1 Detailed Description	1970
0.15.604.2 Constructor & Destructor Documentation	1970
0.15.604.3 Member Function Documentation	1970
0.15.604.4 Field Documentation	1971
0.15.605 Minisat::vec< T > Class Template Reference	1972
0.15.605.1 Constructor & Destructor Documentation	1973
0.15.605.2 Member Function Documentation	1974
0.15.606 smrat::parser::conversion::VectorVariantConverter< Res > Struct Template Reference .	1975
0.15.606.1 Detailed Description	1976
0.15.606.2 Member Typedef Documentation	1976
0.15.606.3 Member Function Documentation	1976
0.15.607 smrat::subtropical::Vertex Struct Reference	1976
0.15.607.1 Detailed Description	1977
0.15.607.2 Constructor & Destructor Documentation	1977
0.15.607.3 Member Function Documentation	1977
0.15.607.4 Field Documentation	1977
0.15.608 smrat::VSModule< Settings > Class Template Reference	1977

0.15.608.1 Member Typedef Documentation	1983
0.15.608.2 Member Enumeration Documentation	1983
0.15.608.3 Constructor & Destructor Documentation	1983
0.15.608.4 Member Function Documentation	1983
0.15.608.5 Field Documentation	2003
0.15.609 Minisat::Watcher Struct Reference	2005
0.15.609.1 Detailed Description	2005
0.15.609.2 Constructor & Destructor Documentation	2005
0.15.609.3 Member Function Documentation	2006
0.15.609.4 Friends And Related Function Documentation	2006
0.15.609.5 Field Documentation	2006
0.15.610 smtrat::ICPModule< Settings >::weights Struct Reference	2006
0.15.610.1 Field Documentation	2006
0.15.611 benchmax::XMLWriter Class Reference	2006
0.15.611.1 Detailed Description	2007
0.15.611.2 Constructor & Destructor Documentation	2007
0.15.611.3 Member Function Documentation	2007
0.15.612 benchmax::Z3 Class Reference	2007
0.15.612.1 Detailed Description	2008
0.15.612.2 Constructor & Destructor Documentation	2008
0.15.612.3 Member Function Documentation	2008
0.15.612.4 Field Documentation	2009

0.1 SMT-RAT

This is the documentation of SMT-RAT, an Open Source C++ Toolbox for Strategic and Parallel SMT Solving. On this page, you can find introductory information on how to obtain and compile SMT-RAT and a traditional doxygen API documentation. The documentation comes in three flavours:

- API documentation as HTML: the regular web-based doxygen documentation is generated by `make doc-apidoc-html` into `build/doc/apidoc-html/` and online at <https://ths-rwth.de/github.io/smtrat>.
- API documentation as PDF: (almost) the full doxygen documentation as a pdf file is generated by `make doc-apidoc-pdf` into `build/doc/doc_smtrat-*.pdf` and online [here](#).
- Manual as PDF: only the manual, suitable for reading as an introduction into SMT-RAT, is generated by `make doc-manual` into `build/doc/manual_smtrat-*.pdf` and online [here](#).

Note that all the information of the manual is contained in the API documentation (both HTML and PDF) as well. It is much more compact, though, and may thus be more approachable as an introduction. However, references to classes do not work in the manual (as the class documentation is not contained).

If you want to use SMT-RAT and want to know how to get and install it, have a look at [Installation](#). It covers the most important steps including obtaining the actual source code, obtaining dependencies, building the library and running our test suite.

If you already use SMT-RAT and want to dig deeper or submit new code, you can additionally browse in [Developers information](#). It contains information about supplementary features like our logging framework and some basic guidelines for our code like how we use doxygen.

Note that this documentation is, and will probably still be for quite some time, work in progress. If you feel that some topic that is important to you is missing or some explanation is unclear, please let us know!

0.2 Installation

0.2.1 Requirements

SMT-RAT is build and tested on Linux. While MacOS is a secondary target (and should work on the most recent version), we do not target Windows yet. Please contact us if you are interested in changing that.

To build and use SMT-RAT, you need the following other software:

- `git` to checkout the git repository.
- `g++` or `clang` to compile.
- `cmake` to generate the make files.

Optional dependencies

- `ccmake` to set `cmake` flags.
- `doxygen` and `doxygen-latex` to build the documentation.

Additionally, SMT-RAT requires a few external libraries, which are installed automatically by CMake if no local version is available:

- `carl` from <http://smtrat.github.io/carl/> (automatically built locally if necessary).
- `boost` (the boost version from CArL is used).

When installing the dependencies, make sure that you meet the following version requirements:

- `g++ >= 9`
- `clang >= 11`

0.2.2 Download

Here are archived versions of SMT-RAT for download:

- [latest](#)

0.2.3 Configuration

SMT-RAT is configured with cmake. To prepare the build and perform the configuration run the following, starting from the root folder of SMT-RAT:

```
$ mkdir build && cd build && cmake ..  
$ ccmake ..
```

ccmake will show the cmake variables. [TODO: document important cmake variables]

0.2.4 Build

To build SMT-RAT use `make` in the build folder:

```
$ make smtrat-shared
```

Relevant targets you may want to build individually include:

- `smtrat-shared`: Builds the (dynamically linked) executable SMT solver
- `smtrat-static`: Builds the (statically linked) executable SMT solver
- `smtrat`: Builds both `smtrat-shared` and `smtrat-static`.
- `doc-html`: Builds the doxygen documentation as HTML.
- `doc-pdf`: Builds the doxygen documentation as PDF.
- `doc`: Builds both `doc-html` and `doc-pdf`.
- `benchmax`: Builds the benchmarking tool.
- `delta`: Builds the delta debugger.

0.2.5 Check build

You can now find an executable `smtrat-shared` in the build directory. It shows some usage information if you run `./smtrat-shared --help`. To run it on an SMT-LIB file, simply run

```
$ ./smtrat-shared example.smt2
```

0.3 System architecture

0.3.1 Different libraries

The different parts of SMT-RAT are split into multiple libraries (in the sense of a shared object library) that are responsible for the following tasks:

- `smtrat-analyzer`: static analysis of input formulae;
- `smtrat-cad`: back all CAD-based techniques;
- `smtrat-common`: common definitions and includes;
- `smtrat-max-smt`: takes care of max SMT queries;
- `smtrat-mcsat`: utilities for the MCSAT-based solver;
- `smtrat-modules`: all regular SMT-RAT modules;
- `smtrat-optimization`: takes care of optimization queries;
- `smtrat-qe`: methods for quantifier elimination;
- `smtrat-solver`: core solving infrastructure;
- `smtrat-strategies`: strategies for SMT solving;
- `smtrat-unsat-cores`: takes care of unsat core computations.

All of these yield a library, while a full-fledged SMT solver is build from the `cli/` path, in particular the `cli/smtratSolver.cpp`.

0.3.2 Software design

The architecture of SMT-RAT puts its focus on modularity and composability of different solving techniques. Every solving technique, for example SAT solving or the simplex method, is encapsulated in a (derivation of the) module class. These modules are composed to a strategy that governs which modules are used in what order. The execution of a strategy is incumbent upon the manager, that also offers an interface for basic SMT solving to the outside.

More advanced solving techniques like quantifier elimination, computing unsatisfiable cores, or tackling max SMT and optimization queries are implemented as individual components in the frontend. The frontend implements (most of) an SMT-LIB compatible interface and can either be used by a generic SMT-LIB parser, or an external tool. This structure is shown in the following picture.



The parser itself is implemented in `cli/parser/` and run from `cli/tools/execute_smtlib.h`. The template argument `Executor` is usually instantiated with the executor from `cli/tools/Executor.h` which corresponds to the frontend. The components in the frontend are taken from the respective SMT-RAT libraries.

The manager is a generic class from `smtrat-solver/Manager.h` that every strategy (from `smtrat-strategies/`) inherits from and only constructs the strategy graph in its constructor. The strategy graph is at the core of the composition of SMT-RAT modules and the following picture shows how a single module is embedded in a strategy.



Every module has (a pointer to) a set of received formulae that represent its input and a set of passed formulae that represent the formula that is passed on to some backend. The module may "solve" the query from its input on its own, or it may pass (one or more) queries to its backends (in this case B-1, B-2 and B-3). Arrows to backends may be labeled with conditions that restrict whether this particular backend can be used, for example checking whether the passed formulae are linear, contain integer variables or bit-vector formulae.

When a module issues a backend call the manager identifies all suitable backend modules (where the condition evaluates to `true`) and calls all backend modules on the passed formulae. This happens either sequentially (until one backend module solves the query) or in parallel.

0.3.3 Modules

A module m holds a set of formulas, called its set of received formulas and denoted by $C_{rcv}(m)$. The main function of a module is `check (bool full)`, which either decides whether $C_{rcv}(m)$ is satisfiable or not, returning SAT or UNSAT, respectively, or returns UNKNOWN. A set of formulas is semantically defined by their conjunction. If the function's argument `full` is set to `false`, the underlying procedure of m is allowed to omit hard obstacles during solving at the cost of returning UNKNOWN in more cases. We can manipulate $C_{rcv}(m)$ by adding (removing) formulas φ to (from) it with `add (\f$\backslash\varphi\f$)` (`remove (\f$\backslash\varphi\f$)`). Usually, $C_{rcv}(m)$ is only slightly changed between two consecutive `check` calls, hence, the solver's performance can be significantly improved if a module works incrementally and supports backtracking. In case m determines the unsatisfiability of $C_{rcv}(m)$, it has to compute at least one preferably small infeasible subset $C_{inf}(m) \subseteq C_{rcv}(m)$. Moreover, a module can specify *lemmas*, which are valid formulas. They encapsulate information which can be extracted from a module's internal state and propagated among other modules. Furthermore, a module itself can ask other modules for the satisfiability of its *set of passed formulas* denoted by $C_{pas}(m)$, if it invokes the procedure `runBackends (bool full)` (controlled by the manager). It thereby delegates work to modules that may be more suitable for the problem at hand.

0.3.4 Strategy

SMT-RAT allows a user to decide how to compose the modules. For this purpose we provide a graphical user interface, where the user can create a *strategy* specifying this composition. A strategy is a directed tree $T := (V, E)$ with a set V of modules as nodes and $E \subseteq V \times \Omega \times \Sigma \times V$, with Ω being the set of *conditions* and Σ being the set of *priority values*. A condition is an arbitrary Boolean combination of formula properties, such as propositions about the Boolean structure of the formula, e.g., whether it is in conjunctive normal form (CNF), about the constraints, \eg whether it contains equations, or about the polynomials, e.g., whether they are linear. Furthermore, each edge carries a unique priority value from $\Sigma = \{1, \dots, |E|\}$.

0.3.5 Manager

The manager holds the strategy and the SMT solver's input formula C_{input} . Initially, the manager calls the method `check` of the module m_r given by the root of the strategy with $C_{rcv}(m_r) = C_{input}$. Whenever a module $m \in V$ calls `runBackends`, the manager adds a solving task (σ, m, m') to its priority queue Q of solving tasks (ordered by the priority value), if there exists an edge $(m, \omega, \sigma, m') \in E$ in the strategy such that ω holds for $C_{pas}(m)$. If a processor p on the machine where SMT-RAT is executed on is available, the first solving task of Q is assigned to p and popped from Q . The manager thereby starts the method `check` of m' with $C_{rcv}(m') = C_{pas}(m)$ and passes the result (including infeasible subsets and lemmas) back to m . The module m can now benefit in its solving and reasoning process from this shared information. Note that a strategy-based composition of modules works incrementally and supports backtracking not just within one module but as a whole. This is realized by a mapping in each module m of its passed formulas $\varphi \in C_{pas}(m)$ to sets $R_1, \dots, R_n \subseteq C_{rcv}(m)$ such that each R_i forms a reason why m included φ in $C_{pas}(m)$ to ask for its satisfiability. In order to exploit the incrementality of the modules, all parallel executed backends terminate in a consistent state (instead of just being killed), if one of them finds an answer.

0.3.6 Procedures implemented as modules

The heart of an SMT solver usually forms a SAT solver. In SMT-RAT, the module `SATModule` abstracts the received formulae to propositional logic and uses the efficient SAT solver `MiniSat` [**minisat**] to find a Boolean assignment of the abstraction. It invokes `runBackends` where the passed formulae of the `SATModule` contain the constraints abstracted by the assigned Boolean variables in a less-lazy fashion [**sebastiani2007lazy**]. [Todo: Make a concise description here, refer to extensive discussion of modules]

0.3.7 Infeasible subsets and lemmas

Infeasible subsets and lemmas, which contain only formulas from $C_{pas}(SATModule)$ of a preceding SATModule, prune its Boolean search space and hence the number of theory calls. Smaller infeasible subsets are usually more advantageous, because they make larger cuts in the search space. We call lemmas containing new constraints inventive lemmas (non-inventive otherwise). They might enlarge the Boolean search space, but they can reduce the complexity of later theory calls. When using inventive lemmas, it is important to ensure that the set possible constraints introduced in such lemmas is finite for a given module and a given input formula. Otherwise, the termination of this procedure cannot be guaranteed. In general, any module might contribute lemmas and all preceding modules in the solving hierarchy can directly involve them in their search for satisfiability.

0.4 Modules

In this chapter we explain how to implement further modules. A module is a derivation of the class `Module` and we give an introduction to its members, interfaces and auxiliary methods in the following of this chapter. A new module and, hence, the corresponding C++ source and header files can be easily created when using the script `writeModules.py`. Its single argument is the module's name and the script creates a new folder in `src/lib/modules/` containing the source and header file with the interfaces yet to implement. Furthermore, it is optional to create the module having a template parameter forming a settings object as explained in sec-:auxfunctions. A new module should be created only this way, as the script takes care of a correct integration of the corresponding code into SMT-RAT. A module can be deleted belatedly by just removing the complete folder it is implemented in.

0.4.1 Main members of a module

Here is an overview of the most important members of the class `Module`.

- `vector<FormulasT> mInfeasibleSubsets`: stores the infeasible subsets of the so far received formulas, if the module determined that their conjunction is not satisfiable.
- `Manager* const mpManager`: a pointer to the manager which maintains the allocation of modules (including this one) to other modules, when they call a backend for a certain formula.
- `const ModuleInput* mpReceivedFormula`: the received formula stores the conjunction of the so far received formulas, which this module considers for a satisfiability check. These formulas are of the type `FormulaT` and the `ModuleInput` is basically a list of such formulas, which never contains a formula more than once.
- `ModuleInput* mpPassedFormula`: the passed formula stores the conjunction of the formulas which this module passes to a backend to be solved for satisfiability. There are dedicated methods to change this member, which are explained in the following.

The received formula of a module is the passed formula of the preceding module. The owner is the preceding module, hence, a module has only read access to its received formula. The `ModuleInput` also stores a mapping of a sub-formula in the passed formula of a module to its origins in the received formula of the same module. Why this mapping is essential and how we can construct it is explained in Section sec:runbackend.

0.4.2 Interfaces to implement

In the following we explain which methods must be implemented in order to fill the module's interfaces with life. All these methods are the core implementation and wrapped by the actual interfaces. This way the developer of a new module needs only to take care about the implementation of the actual procedure for the satisfiability check. All infrastructure-related actions are performed by the actual interface.

0.4.2.1 Informing about a constraint

```
bool MyModule::informCore( const Formula& _constraint )
{
    // Write the implementation here.
}
```

Informs the module about the existence of the given constraint (actually it is a formula wrapping a constraint) usually before it is actually added to this module for consideration of a later satisfiability check. At least it can be expected, that this method is called, before a formula containing the given constraint is added to this module for consideration of a later satisfiability check. This information might be useful for the module, e.g., for the initialization of the data structures it uses. If the module can already decide whether the given constraint is not satisfiable itself, it returns `false` otherwise `true`.

0.4.2.2 Adding a received formula

```
bool MyModule::addCore( const ModuleInput::const_iterator )
{
    // Write the implementation here.
}
```

Adds the formula at the given position in the conjunction of received formulas, meaning that this module has to include this formula in the next satisfiability check. If the module can already decide (with very low effort) whether the given formula is not satisfiable in combination with the already received formulas, it returns `false` otherwise `true`. This is usually determined using the solving results this module has stored after the last consistency checks. In the most cases the implementation of a new module needs some initialization in this method.

0.4.2.3 Removing a received formula

```
void MyModule::removeCore( const ModuleInput::iterator )
{
    // Write the implementation here.
}
```

Removes the formula at the given position from the received formula. Everything, which has been stored in this module and depends on this formula must be removed.

0.4.2.4 Checking for satisfiability

```
Answer MyModule::checkCore( bool )
{
    // Write the implementation here.
}
```

Implements the actual satisfiability check of the conjunction of formulas, which are in the received formula. There are three options how this module can answer: it either determines that the received formula is satisfiable and returns `true`, it determines unsatisfiability and returns `false`, or it cannot give a conclusive answer and returns `UNKNOWN`. A module has also the opportunity to reason about the conflicts occurred, if it determines unsatisfiability. For this purpose it has to store at least one infeasible subset of the set of so far received formulas. If the method `check` is called with its argument being `false`, this module is allowed to omit hard obstacles during solving at the cost of returning `UNKNOWN` in more cases, we refer to as a *lightweight check*.

0.4.2.5 Updating the model/satisfying assignment

```
void MyModule::updateModel()
{
    // Write the implementation here.
}
```

If this method is called, the last result of a satisfiability check was `true` and no further formulas have been added to the received formula, this module needs to fill its member `mModel` with a model. This model must be complete, that is all variables and uninterpreted functions occurring in the received formula must be assigned to a value of their corresponding domain. It might be necessary to involve the backends using the method `getBackendsModel()` (if they have been asked for the satisfiability of a sub-problem). It stores the model of one backend into the model of this module.

0.4.3 Running backend modules

Modules can always call a backend in order to check the satisfiability of any conjunction of formulas. Fortunately, there is no need to manage the assertion of formulas to or removing of formulas from the backend. This would be even more involved as we do allow changing the backend if it is appropriate (more details to this are explained in Chapter [chapter:composingats](#)). Running the backend is done in two steps:

1. Change the passed formula to the formula which should be solved by the backend. Keep in mind, that the passed formula could still contain formulas of the previous backend call.
2. Call `runBackends(full)`, where `full` being `false` means that the backends have to perform a lightweight check.

The first step is a bit more tricky, as we need to know which received formulas led to a passed formula. For this purpose the `ModuleInput` maintains a mapping from a passed sub-formula to one or more conjunctions of received sub-formulas. We give a small example. Let us assume that a module has so far received the following constraints (wrapped in formulas)

$$c_0 : x \leq 0, c_1 : x \geq 0, c_2 : x = 0$$

and combines the first two constraints c_0 and c_1 to c_2 . Afterwards it calls its backend on the only remaining constraint, that means the passed formula contains only $c_2 : x = 0$. The mapping of c_2 in the passed formula to the received sub-formulas it stems from then is

$$c_2 \mapsto (c_0 \wedge c_1, c_2).$$

The mapping is maintained automatically and offers two methods to add formulas to the passed formulas:

```
pair<ModuleInput::iterator,bool> addReceivedSubformulaToPassedFormula(ModuleInput::const_iterator)
```

Adds the formula at the given position in the received formula to the passed formulas. The mapping to its *original formulas* contains only the set consisting of the formula at the given position in the received formula.

```
pair<ModuleInput::iterator,bool> addSubformulaToPassedFormula(const Formula&)
pair<ModuleInput::iterator,bool> addSubformulaToPassedFormula(const Formula&, const Formula&)
pair<ModuleInput::iterator,bool> addSubformulaToPassedFormula(const Formula&, shared_ptr<vector<FormulaT>>&)
```

Adds the given formula to the passed formulas. It is mapped to the given conjunctions of origins in the received formula. The second argument (if it exists) must only consist of formulas in the received formula.

It returns a pair of a position in the passed formula and a `bool`. The `bool` is `true`, if the formula at the given position in the received formula has been added to the passed formula, which is only the case, if this formula was not yet part of the passed formula. Otherwise, the `bool` is `false`. The returned position in the passed formula points to the just added formula.

The vector of conjunctions of origins can be passed as a shared pointer, which is due to a more efficient manipulation of these origins. Some of the current module implementations directly change this vector and thereby achieve directly a change in the origins of a passed formula. \end{itemize} If, by reason of a later removing of received formulas, there is no conjunction of original formulas of a passed formula left (empty conjunction are removed), this passed formula will be automatically removed from the backends and the passed formula. That does also mean, that if we add a formula to the passed formula without giving any origin (which is done by the first version of `addSubformulaToPassedFormula`), the next call of `removeSubformula` of this module removes this formula from the passed formula. Specifying received formulas being the origins of a passed formula highly improves the incremental solving performance, so we recommend to do so.

The second step is really just calling `runBackends` and processing its return value, which can be `True`, `False`, or `Unknown`.

0.4.4 Auxiliary functions

The Module class provides a rich set of methods for the analysis of the implemented procedures in a module and debugging purposes. Besides all the printing methods, which print the contents of a member of this module to the given output stream, SMT-RAT helps to maintain the correctness of new modules during their development. It therefore provides methods to store formulas with their assumed satisfiability status in order to check them belatedly by any SMT solver which is capable to parse `.smt2` files and solve the stored formula. To be able to use the following methods, the compiler flag `SMT RAT_DEVOPTION_Validation` must be activated, which can be easily achieved when using, e.g., `ccmake`.

- `void checkInfeasibleSubsetForMinimality(vector<FormulasT>::const_iterator, const string&, unsigned)` const This method checks the infeasible subset at the given position for minimality, that is it checks whether there is a subset of it having maximally \$n\$ elements less while still being infeasible. As for some approaches it is computationally too hard to provide always a minimal infeasible subset, they rather provide infeasible subsets not necessarily being minimal. This method helps to analyze how close the size of the encountered infeasible subsets is to a minimal one.
- Another important feature during the development of a new module is the collection of statistics. The script `writeModules.py` for the creation of a new module automatically adds a class to maintain statistics in the same folder in which the module itself is located. The members of this class store the statistics usually represented by primitive data types as integers and floats. They can be extended as one pleases and be manipulated by methods, which have also to be implemented in this class. SMT-RAT collects and prints these statistics automatically, if its command line interface is called with the option `--statistics` or `-s`.
- If the script `writeModules.py` for the creation of a new module is called with the option `-s`, the module has also a template parameter being a `Settings` object. The different `Settings` objects are stored in the `Settings` file again in the same folder as the module is located. Each of these setting objects assigns all settings, which are usually of type `bool`, to values. The name of these objects must be of the form `XY-SettingsN`, if the module is called `XYModule` and with `N` being preferably a positive integer. Fulfilling these requirements, the settings to compile this module with, can be chosen, e.g. with `ccmake`, by setting the compiler flag `SMT RAT_XY_Settings` to `N`. Within the implementation of the module, its settings can then be accessed using its template parameter `Settings`. If, for instance, we want to change the control flow of the implemented procedure in the new module depending on a setting `mySetting` being `true`, we write the following:

```
..  
if(Settings::mySettings)  
{  
    ..  
}  
..
```

This methodology assures that the right control flow is chosen during compilation and, hence, before runtime.

SMT-RAT contributes a toolbox for composing an SMT compliant solver for its supported logics, that means it is incremental, supports backtracking and provides reasons for inconsistency. The resulting solver is either a fully operative SMT solver, which can be applied directly on .smt2 files, or a theory solver, which can be embedded into an SMT solver in order to extend its supported logics by those provided by SMT-RAT.

We are talking about composition and toolbox, as SMT-RAT contains implementations of many different procedures to tackle, \eg \supportedLogics, each of them embedded in a module with uniform interfaces. These modules form the tools in the toolbox and it is dedicated to a user how to use them for solving an SMT formula.

In Section sec::strategy we have already introduced a strategy and in the following of this chapter we give a brief introduction to the existing modules equipped with an estimation of their input-based performances.

0.4.5 Existing module implementation

[Available modules](#)

0.4.6 Available modules

SMT-RAT comes with the following modules that are documented in some more detail below:

BEModule BVMODULE CNFerModule CSplitModule CoCoAGBModule CubeLIAModule CurryModule EMModule
ESModule FPPModule FouMoModule GBModule GBPPModule ICEModule ICPModule IncWidthModule
IntBlastModule IntEqModule LRAModule LVEModule MCBModule NRAILModule NewCADModule NewCoveringModule
PBGaussModule PBPPModule PFEModule SATModule STropModule SplitSOSModule SymmetryModule
UFCegarModule UnionFindModule VSModule

0.4.6.1 BEModule

0.4.6.2 BVMODULE

This module implements an encoder for bitvector formulae to propositional logic. It is described in more detail in [?.](#)

0.4.6.3 CNFerModule

Transforms its received formula into conjunctive normal form CNF.

0.4.6.3.0.1 Efficiency The worst case complexity of this module is polynomial in the number of operators in the formula to transform.

0.4.6.4 CSplitModule

Implements solving for nonlinear integer arithmetic using incremental linearization. This module was implemented in [?](#) based on [?](#).

0.4.6.5 CoCoAGBModule

Uses Gröbner bases for theory solving, very much like [smrat::GBModule](#). However, for the underlying implementation of Gröbner bases, this module does not use `carl::groebner` but `CoCoALib`.

0.4.6.6 CubeLIAModule

Implements cube-based tests for linear integer arithmetic based on [?](#).

0.4.6.7 CurryModule

Implements curryfying preprcessing as described in [?](#).

0.4.6.8 EMModule

This module addempts to eliminate multiple factors from equations and inequations.

0.4.6.9 ESModule

Uses equations (or Boolean assertions) to eliminate variables from the remaining formula. Let the formula have the form $e \wedge \varphi'$, then we use knowledge gained from e to simplify φ' . If e is an arithmetic equation such that we can rewrite it to the form $x = t$ (with x a variable) then we substitute t for x into φ' . Otherwise we simply replace e with `true` in φ' . This is done recursively in the formula.

0.4.6.10 FPPModule

This module implements a generic preprocessing facility. It runs a given strategy on the input and retrieves the simplified formula using the facilities of the [Manager](#) class. This process is iterated until a fix point is reached – no further simplification was done by the strategy – or a predefined number of iterations was performed. The resulting simplified formula is then forwarded to the backend.

The strategy should be a linear sequence of preprocessing modules, as passing simplified formulas back to the caller is not possible in a meaningful way for general [Module](#) classes.

0.4.6.11 FouMoModule

Implements the SMT compliant Fourier-Motzkin algorithm. Hence, this module can decide the consistency of any conjunction consisting only of linear real arithmetic constraints. Furthermore, it might also find the consistency of a conjunction of constraints even if they are not all linear e.g. in the case of a monomial x^i (where i is a positive integer) only occuring in the shape of this monomial. Such a monomial is subsequently eliminated as common linear variables are eliminated. One can tune a threshold parameter in order to determine when, regarding the size of the considered constraints, this module shall call the backends. In the latter case, the backends are called with the constraint set that is obtained after eliminating a certain number of variables.

0.4.6.11.1 Integer arithmetic One can also use this approach for (linear) integer arithmetic as unsatisfiability over the real domain implies unsatisfiability over the integer domain. In addition to that, one can heuristically try to construct integer solutions by considering the lowest upper and the highest lower bound of a variable that can be derived from the respective elimination step. Note that this approach is incomplete.

0.4.6.12 GBModule

Implements the Gröbner bases based procedure as presented in ?. In general, this procedure can detect only the unsatisfiability of a conjunction of equations. This module also supports the usage of these equations to further simplify all constraints in the conjunction of constraints forming its input and passes these simplified constraints to its backends. However, it cannot be guaranteed that backends perform better on the simplified constraints than on the constraints before simplification.

0.4.6.12.0.1 Efficiency The worst case complexity of the underlying procedure is exponential in the number of variables of the input constraints. In the case that the conjunction of constraints to check for satisfiability contains equations, this module can be more efficient than other modules for NRA on finding out inconsistency.

0.4.6.13 GBPPModule

Uses Gröbner basis computations to simplify the input formula. The underlying implementation of Gröbner bases is used from `carl::groebner`.

The fundamental idea is as follows: Separate the input formula into equalities and inequalities and compute the Gröbner basis of the equalities. Now we can replace the equalities with the Gröbner basis (if this "seems easier") and also simplify the inequalities using the Gröbner basis.

0.4.6.14 ICEModule

This module tries to find simple chains of inequalities that can be combined to form a cycle. It converts constraints of the form $x \geq y + z$ to a hypergraph (with an edge going from x to y, z) and uses the coefficients as edge weights. If this hypergraph contains cycles, these can be used to infer additional constraints on the individual variables. In particular, zero cycles induce equality of variables while negative cycles reveal conflicts.

0.4.6.15 ICPModule

Implements a combination of interval constraint propagation equipped with a Newton-based contraction ? and LRA solving, for which we use our LRAModule. The implementation is inspired by ?, but additionally interacts with backends in order to exploit their efficiency on examples, where ICP fails. It thereby incorporates the possibility to invoke lightweight checks and highly benefits from the backends being optimized for small domains as, e.g. described in ?. This module tries to lift splitting decisions as well as lemmas encoding a nonzero contraction to a preceding SATModule. It ensures an efficient processing of these decisions, which are this way shared with other modules.

0.4.6.15.0.1 Efficiency It is very difficult to give a conclusive statement about the efficiency of ICP. Usually, it performs better, if the domains of all variables are bounded intervals, preferably with a small diameter. It might also benefit from a higher number of constraints, as this introduces more chances for the propagation. However, more constraints mean also more overhead.

0.4.6.16 IncWidthModule

This module is meant to be used for solving nonlinear integer arithmetic problems by encoding them into bitvector arithmetic formulas (as done in [smrat::IntBlastModule](#)), as described in [?](#) and [?](#), both (heavily) inspired by [?](#). This approach heavily benefits from knowing bounds on individual variables, both in terms of its ability to find infeasibility and its performance in general.

In this module, we use [smrat::ICPModule](#) to infer new bounds and call the backend incrementally with growing bounds until the computed upper bounds are met.

0.4.6.17 IntBlastModule

This module implements encoding nonlinear integer arithmetic problems into bitvector arithmetic formulas as described in [?](#) and [?](#), both (heavily) inspired by [?](#).

0.4.6.18 IntEqModule

Implements the SMT compliant equation elimination method presented in [?](#). Hence, this module either determines that the equations of an instance are unsatisfiable or calculates a certain number of substitutions for variables. The latter can be used to eliminate the equations by substituting every variable for which a substitution exists by its substitution. The set of constraints that we obtain with this procedure, not containing equations anymore, is passed to the backends.

0.4.6.19 LRAModule

Implements the SMT compliant simplex method presented in [?](#). Hence, this module can decide the consistency of any conjunction consisting only of linear real arithmetic constraints. Furthermore, it might also find the consistency of a conjunction of constraints even if they are not all linear and calls a backend after removing some redundant linear constraints, if the linear constraints are satisfiable and the found solution does not satisfy the non-linear constraints. Note that the [smrat::LRA Module](#) might need to communicate a lemma/tautology to a preceding [smrat::SATModule](#), if it receives a constraint with the relation symbol \neq and the strategy needs for this reason to define a [smrat::SATModule](#) at any position before an [smrat::LRA Module](#).

0.4.6.19.0.1 Integer arithmetic In order to find integer solutions, this module applies, depending on which settings are used, branch-and-bound, the construction of Gomory cuts and the generation of cuts from proofs [?](#). It is also supported to combine these approaches. Note that for all of them the [smrat::LRA Module](#) needs to communicate a lemma/tautology to a preceding SATModule and the strategy needs for this reason to define a SATModule at any position before an [smrat::LRA Module](#).

0.4.6.19.0.2 Efficiency The worst case complexity of the implemented approach is exponential in the number of variables occurring in the problem to solve. However, in practice, it performs much faster, and the worst case applies only on very artificial examples. This module outperforms any module implementing a method that is designed for solving formulas with non-linear constraints. If the received formula contains integer valued variables, the aforementioned methods might not terminate.

0.4.6.20 LVEModule

This module aims to eliminate lone variables that only occur once. We call a variable a lone variable if it only occurs in a single constraint. Furthermore we only try to eliminate the variable if this constraint is a top-level constraint.

0.4.6.21 MCBModule

This module attempts to detect logic structures that essentially encode a multiple-choice on an arithmetic variable. It extracts this information and translates it to a Boolean structure, thereby eliminating the arithmetic variable.

0.4.6.22 NRAILModule

This module implements incremental linearization as described in ?, roughly following ? and ?. In addition, it implements an ICP-based axiom instantiation.

0.4.6.23 NewCADModule

This module is based on the implementation in [smtrat::cad](#). It is described in ? in some detail.

0.4.6.24 PBGaussModule

This module tries to simplify pseudo-Boolean problems using Gauss as described in ?.

0.4.6.25 PBPPModule

This module implements a variety of preprocessing techniques for pseudo-Boolean problems. Many of them are based on ?.

0.4.6.26 PFEModule

Implements factor elimination in polynomials based on factorization and variable bounds.

Given a constraint $p \sim 0$ with $\sim \in \{=, \neq, \geq, >, \leq, <\}$ and bounds b on the variables in p , the module does the following: It computes a factor q such that $p = q \cdot r$ and evaluates q on the intervals represented by the bounds. If the resulting interval $q(b)$ is sign-invariant, the constraint can be simplified or additional constraints can be added. We consider an interval to be sign-invariant, if it is positive, semi-positive, zero, semi-negative or negative.

0.4.6.26.0.1 Simplifications The following simplifications are done for for $q \cdot r \sim 0$:

\sim	$q(b) > 0$	$q(b) \geq 0$	$q(b) = 0$	$q(b) < 0$	$q(b) \leq 0$
$=$	$p := r$	$f := q = 0 \vee r = 0$	$p := 0$	$p := r$	$f := q = 0 \vee r = 0$
\neq	$p := r$	$f := q > 0 \wedge r \neq 0$	$p := 0$	$p := r$	$f := q < 0 \wedge r \neq 0$
\geq	$p := r$	$f := q = 0 \vee r \geq 0$	$p := 0$	$c := r \leq 0$	$f := q = 0 \vee r \leq 0$
$>$	$p := r$	$f := q > 0 \wedge r > 0$	$p := 0$	$c := r < 0$	$f := q < 0 \wedge r < 0$
\leq	$p := r$	$f := q = 0 \vee r \leq 0$	$p := 0$	$c := r \geq 0$	$f := q = 0 \vee r \geq 0$
$<$	$p := r$	$f := q > 0 \wedge r < 0$	$p := 0$	$c := r > 0$	$f := q < 0 \wedge r > 0$

Notation: $p := p'$ replaces the polynomial, $c := p' \sim 0$ replaces the whole constraint by a new one and $f := f'$ replaces the constraint by a new formula.

To maximize the cases where $q(b)$ actually is sign-invariant, we proceed as follows. We compute a factorization of p , that is a number of polynomials p_i such that $p = \prod_{i=1}^k p_i$. We now separate all p_i into two sets: P_q for all sign-invariant p_i and P_r for all other p_i . Thereby, we set $q = \prod_{t \in P_q} t$ and $r = \prod_{t \in P_r}$ and know that $q(b)$ is sign-invariant.

Instead of computing $q(b)$ once again afterwards, we can determine the type of sign-invariance of $q(b)$ from the types of sign-invariances of the factors from P_q . Assume that we start with a canonical factor 1 and a sign-invariance of $>$, we can iteratively combine them like this:

0.4.6.26.0.2 Combining types of sign-invariance Combining two types of sign-invariance is done as follows:

.	>	\geq	=	\leq	<
>	>	\geq	=	\leq	<
\geq	\geq	\geq	=	\leq	\leq
=	=	=	=	=	=
\leq	\leq	\leq	=	\geq	\geq
<	<	\leq	=	\geq	>

0.4.6.27 SATModule

This module abstracts it's received formula, being any SMT formula of the supported logics of \smrat, to it's Boolean skeleton. It thereby replaces all constraints in the formula by fresh Boolean variables. The resulting propositional formula is then solved with \minisat~[where] after each completed decision level the constraints belonging to the assigned Boolean variables are checked for consistency by the backends of this module. In the case of inconsistency, the infeasible subsets of the backends are abstracted and then involved in the search for a satisfying solution.

0.4.6.27.0.1 Efficiency Even though the worst case complexity of this procedure, not considering the complexity of the backend calls, is exponential in the number of variables in the abstracted formula, the procedure is in practice more efficient than any of the theory modules. Hence, it does clearly not form a bottleneck of the SMT solving. However, one should aim at reducing the number and complexity of the theory (backend) calls of this module, which might be influenced by infeasible subsets, which are small and/or involve constraints of earlier decision levels in the SAT solving, and lemmas, which either prune the search space of the SAT solving or ease subsequent theory calls.

0.4.6.28 STropModule

Implements the subtropical satisfiability method as described in ? based on ?.

0.4.6.29 SplitSOSModule

Splits the left hand side of constraints according to their sum-of-squares decomposition.

0.4.6.30 SymmetryModule

This module tries to recognize syntactic symmetries in the input formula and adds symmetry breaking constraints. The core functionality is provided by CArL through carl::formula::breakSymmetries() which internally encodes the formula as a graph and uses bliss to find automorphisms on this graph.

0.4.6.30.0.1 Efficiency Finding automorphisms is as difficult as determining whether two graphs are isomorphic, and it is not known whether this problem can be solved in polynomial or exponential time. In practice, current solvers like bliss perform very good on large graphs and we therefore assume this module to be sufficiently fast.

0.4.6.31 UFCegarModule

This module implements a CEGAR-approach for solving uninterpreted functions (or rather reducing uninterpreted functions to equality logic).

0.4.6.32 UnionFindModule

This module implements a theory solver for equality logic based on a persistent union find structure (based on the immer library).

0.4.6.33 VSModule

Implements the virtual substitution method for a conjunction of constraints as described in ?. This module supports incremental calls, efficient backtracking and infeasible subset generation. Note, that the infeasible subsets are often very small but not necessarily minimal. The implemented approach is not complete, as it maybe cannot decide the satisfiability of a conjunction containing a constraint, which involves a variable with degree \$3\$ or more. Note, that even if no constraint of such form occurs in the received formula, this module might not be able to determine the consistency of its received formula, as it could create constraints of this form in its solving process. Nevertheless, the implemented approach is efficient compared to other approaches for non-linear real arithmetic conjunctions, and therefore well-suited to be used for solving conjunctions of non-linear real arithmetic constraints before complete approaches have their try. In combination with a backend, this module tries to solve the given problem and calls the backend on problems with less variables.

0.4.6.33.0.1 Efficiency The worst case complexity of this approach is exponential in the number of real arithmetic variables occurring in the conjunction to solve. It performs especially good on almost linear instances and slightly prefers problems only containing constraints with the relation symbols \leq , \geq and $=$. It is often the case, that even if the conjunction to solve contains many not suited constraints, this module can determine the consistency on the basis of a well suited subset of the constraints in this conjunction.

0.5 Strategies

To compose modules to a solver, strategies are used. Strategies are represented as trees of modules where the input file is passed to the root module. Every module can issue calls to its **backend modules** (via `runBackends`) which then calls its child modules in the strategy on the formula this module has put in the passed formulas. If no backend module exists, the call returns `unknown`.

A Strategy is specified by deriving from the Manager class and settings its strategy in the constructor using the `setStrategy` member function. A simple example, supporting only propositional logic, would look like this:

```
class PureSAT: public Manager {
public:
    PureSAT(): Manager() {
        setStrategy(
            addBackend<SATModule<SATSettings1>>()
        );
    }
};
```

Each module is added to the strategy tree using `addBackend<Module class>` where the module class is usually a template that is instantiated with some settings. A slightly more complex, where the SATModule has the NewCADModule available for theory calls, might look as follows:

```
class CADOnly: public Manager {
public:
    CADOnly(): Manager() {
        setStrategy(
            addBackend<SATModule<SATSettings1>>(
                addBackend<NewCADModule<NewCADSettingsFOS>>()
            )
        );
    }
};
```

Note that the arguments to `setStrategy` and `addBackend` can not only be a single element, but an `initializer_list` of multiple backends as well. In this case the backends are called in order until one returns a conclusive result (that is: not `unknown`). If SMT-RAT was configured to allow for parallel execution, the backends are executed in parallel and the result of the first backend to terminate is used.

```
class CADVSICP: public Manager {
public:
    CADVSICP(): Manager() {
        setStrategy(
            addBackend<SATModule<SATSettings1>>({
                addBackend<NewCADModule<NewCADSettingsFOS>>(
                    addBackend<VSMModule<VSSettings234>>()
                ),
                addBackend<ICPModule<ICPSettings1>>()
            })
        );
    }
};
```

Finally, we can also attach conditions to individual backends to restrict their applicability. For example, we can construct a strategy that uses separate sub-strategies for linear and nonlinear problems:

```
class RealSolver: public Manager {
    static bool is_nonlinear(carl::Condition condition) {
        return (carl::PROP_CONTAINS_NONLINEAR_POLYNOMIAL <= condition);
    }
    static bool is_linear(carl::Condition condition) {
        return !(carl::PROP_CONTAINS_NONLINEAR_POLYNOMIAL <= condition);
    }
public:
    RealSolver(): Manager() {
        setStrategy({
            addBackend<FPPModule<FPPSettings1>>(
                addBackend<SATModule<SATSettings1>>(
                    addBackend<ICPModule<ICPSettings1>>(
                        addBackend<VSMModule<VSSettings234>>(
                            addBackend<NewCADModule<NewCADSettingsFOS>>()
                        )
                    )
                )
            )
        }).condition( &is_nonlinear ),
        addBackend<FPPModule<FPPSettings1>>(
            addBackend<SATModule<SATSettings1>>(
                addBackend<LRAModule<LRASettings1>>()
            )
        ).condition( &is_linear )
    });
};
```

0.6 Using SMT-RAT

SMT-RAT can be used in two ways, either as a standalone solver or as a C++ library within some other software.

0.6.1 Standalone solver

Before actually compiling SMT-RAT into a binary, an appropriate strategy should be selected. While a number of strategies are available, it sometimes makes sense to craft a strategy for the specific problem at hand. Please refer to [Strategies](#) on how to create strategies. To select a strategy use `ccmake` to set the `SMT RAT_Strategy` variable accordingly and then build the solver binary using `make smtrat-shared`. Use `./smtrat-shared --strategy` to check whether the desired strategy is used.

The solver binary can now be used to solve input in the `smt2` format, either given as a filename or on standard input:

```
./smtrat-shared <input file>
cat <input file> | ./smtrat-shared -
```

Note that the solver binary can perform many other tasks as well that we discuss below. Some of these are enabled (or disabled) by a set of `cmake` options of the form `CLI_ENABLE_*` and the currently available ones can be obtained as follows:

```
$ ./smtrat-shared --help
Usage: ./smtrat-shared [options] input-file

Core settings:
  --help                                show help
  --info                                 show some basic information about this
                                         binary
  --version                             show the version of this binary
  --settings                            show the settings that are used
  --cmake-options                      show the cmake options during
                                         compilation
  --strategy                           show the configured strategy
  --license                            show the license
  -c [ --config ] arg                  load config from the given config file

Solver settings:
  --preprocess                          only preprocess the input
  --pp-output-file arg                 store the preprocessed input to this
                                         file
  --to-cnf-dimacs                     transform formula to cnf as dimacs
  --to-cnf-smtlib                      transform formula to cnf as smtlib
  --print-model                         print a model if the input is
                                         satisfiable
  --print-all-models                   print all models of the input

Validation settings:
  --validation.export-smtlib          store validation formulas to smtlib
                                         file
  --validation.smtlib-filename arg   (=validation.smt2)
                                         filename of smtlib output
  --validation.channel arg            add a channel to be considered

Module settings:
  --module.parameter arg              add a parameter for modules (key=value)

Parser settings:
  --dimacs                            parse input file as dimacs file
  --opb                               parse input file as OPB file
  --input-file arg                    path of the input file
  --disable-uf-flattening           disable flattening of nested
                                         uninterpreted functions
  --disable-theory                   disable theory construction

Analysis settings:
  --analyze.enabled                   enable formula analyzer
  --analyze.projections arg          (=none)
                                         which CAD projections to analyze (all,
                                         collins, hong, mccallum,
                                         mccallum_partial, lazar, brown, none)

CAD Preprocessor settings:
  --cad.pp.no-elimination           disable variable elimination
  --cad.pp.no-resultants            disable resultant rule
```

0.6.1.1 Formula analysis

One sometimes wants to only obtain certain information about the given formula, usually for statistical purposes. SMT-RAT exposes a formula analyzer that gives a number of properties of the formula.

```
$ ./smtrat-shared --analyze.enabled <input file>
```

0.6.1.2 Preprocessing

While many SMT-RAT strategies employ certain preprocessing techniques, it is sometimes convenient to apply this preprocessing ahead of time, for example to normalize the inputs. The result is either printed or written to an output file if `--pp-output-file` is given.

```
$ ./smtrat-shared --preprocess --pp-output-file <output file> <input file>
```

0.6.1.3 Quantifier elimination

NOTE: This feature is currently not maintained. Expect bugs and wrong answers. This feature is disabled in the default build of SMT-RAT.

Instead of regular SMT solving, SMT-RAT can also perform quantifier elimination tasks as described in [Neuhaeuser2018]. This technique is used when the SMTLIB file contains a `eliminate-quantifiers` command like `(eliminate-quantifiers (exists x y) (forall z))`.

0.6.1.4 DIMACS solving

For purely propositional formulae (i.e. SAT solving) one usually uses the (much more compact) DIMACS format instead of SMT-LIB. Note that SMT-RAT still uses the configured strategy to solve the given input instead of only a SAT solver, allowing to use custom preprocessing techniques.

```
$ ./smtrat-shared --dimacs <input file>
```

0.6.1.5 Pseudo-Boolean solving

Another interesting solving task is concerned with pseudo-Boolean formulae where Boolean variables are used in arithmetic constraints (and implicitly considered as being integers over just zero and one). A special input format called OPB exists (rather similar to DIMACS) that can be read and solved with a strategy based on `? as follows:`

```
$ ./smtrat-shared --opb <input file>
```

0.6.1.6 Optimization

For many applications, one wants not only some feasible solution but rather an optimal solution with respect to some objective function. This is used when the SMTLIB file contains one or more `(minimize)` or `(maximize)` commands with semantics similar to what is described for [z3](#).

0.6.2 Embedding in other software

Instead of using SMT-RAT as a standalone solver, it can also be embedded in other software.

0.6.2.1 Interface

The easiest way is to embed a certain strategy.

If for instance the SMT solver based on the strategy of RatOne shall be used (we can also choose any self-composed strategy here), we can create it as follows:

```
smtrat::RatOne yourSolver = smtrat::RatOne();
```

As all strategies are derived from the Manager class, we can thus use the Manager interface. The most important methods are the following:

- `bool inform(const FormulaT&)` Informs the solver about a constraint, wrapped by the given formula. Optimally, the solver should be informed about all constraints, which it will receive eventually, before any of them is added as part of a formula with the interface `add(...)`. The method returns `false` if it is easy to decide (for any module used in this solver), whether the constraint itself is inconsistent.
- `bool add(const FormulaT&)` Adds the given formula to the conjunction of formulas, which will be considered for the next satisfiability check. The method returns `false`, if it is easy to decide whether the just added formula is not satisfiable in the context of the already added formulas. Note, that only a very superficial and cheap satisfiability check is performed and mainly depends on solutions of previous consistency checks. In the most cases this method returns `true`, but in the case it does not the corresponding infeasible subset(s) can be obtained by `infeasibleSubsets()`.
- `Answer check(bool)` This method checks the so far added formulas for satisfiability. If, for instance we extend an SMT solver by a theory solver composed with SMT-RAT, these formulas are only constraints. The answer can either be `SAT`, if satisfiability has been detected, or `UNSAT`, if the formulas are not satisfiable, and `UNKNOWN`, if the composition cannot give a conclusive answer. If the answer has been `SAT`, we get the model, satisfying the conjunction of the given formulas, using `model()` and, if it has been `UNSAT`, we can obtain infeasible subsets by `infeasibleSubsets()`. If the answer is `UNKNOWN`, the composed solver is either incomplete (which highly depends on the strategy but for `QF_NRA` it is actually always possible to define a strategy for a complete SMT-RAT solver) or it communicates lemmas/tautologies, which can be obtained applying `lemmas()`. If we embed, e.g., a theory solver composed with SMT-RAT into an SMT solver, these lemmas can be used in its sat solving process in the same way as infeasible subsets are used. The strategy of an SMT solver composed with SMT-RAT has to involve a `SATModule` before any theory module is used (It is possible to define a strategy using conditions in a way, that we achieve an SMT solver, even if for some cases no `SATModule` is involved before a theory module is applied.) and, therefore, the SMT solver never communicates these lemmas as they are already processed by the `SATModule`. A better explanation on the modules and the strategy can be found in [system architecture](#). If the Boolean argument of the function `check` is `false`, the composed solver is allowed to omit hard obstacles during solving at the cost of returning `UNKNOWN` in more cases.
- `void push()` Pushes a backtrack point to the stack of backtrack points.
- `bool pop()` Pops a backtrack point from the stack of backtrack points and undoes everything which has been done after adding that backtrack point. It returns `false` if no backtrack point is on the stack. Note, that SMT-RAT supports incrementality, that means, that by removing everything which has been done after adding a backtrack point, we mean, that all intermediate solving results which only depend on the formulas to remove are deleted. It is highly recommended not to remove anything, which is going to be added directly afterwards.
- `const std::vector<FormulasT>& infeasibleSubsets() const` Returns one or more reasons for the unsatisfiability of the considered conjunction of formulas of this SMT-RAT composition. A reason is an infeasible subset of the sub-formulas of this conjunction.
- `const Model& model() const` Returns an assignment of the variables, which occur in the so far added formulas, to values of their domains, such that it satisfies the conjunction of these formulas. Note, that an assignment is only provided if the conjunction of so far added formulas is satisfiable. Furthermore, when solving non-linear real arithmetic formulas the assignment could contain other variables or freshly introduced variables.

- `std::vector<FormulaT> lemmas()` const Returns valid formulas, which we call lemmas. For instance the `ICPModule` might return lemmas being splitting decisions, which need to be processed in, e.g., a SAT solver. A *splitting decision* has in general the form $(c_1 \text{ and } \dots \text{ and } c_n) \rightarrow (p \leq r \text{ or } p > r)$ where c_1, \dots, c_n are constraints of the set of currently being checked constraints (forming a *premise*), p is a polynomial (in the most cases consisting only of one variable) and r being a rational number. Hence, splitting decisions always form a tautology. We recommend to use the `ICPModule` only in strategies with a preceding `SATModule`. The same holds for the `LRAModule`, `VSMModule`, and `CADModule` if used on QF_NIA formulas. Here, again, splitting decisions might be communicated.

Of course, the Manager interface contains more methods that can be found at Manager.

0.6.2.2 Syntax of formulas

The class `Formula` represents SMT formulas, which are defined according to the following abstract grammar

$$\begin{array}{lcl}
p & ::= & a \quad | \quad b \quad | \quad x \quad | \quad (p + p) \quad | \quad (p \cdot p) \quad | \quad (p^e) \\
v & ::= & u \quad | \quad x \\
s & ::= & f(v, \dots, v) \quad | \quad u \quad | \quad x \\
e & ::= & s = s \\
c & ::= & p = 0 \quad | \quad p < 0 \quad | \quad p \leq 0 \quad | \quad p > 0 \quad | \quad p \geq 0 \quad | \quad p \neq 0 \\
\varphi & ::= & c \quad | \quad (\neg\varphi) \quad | \quad (\varphi \wedge \varphi) \quad | \quad (\varphi \vee \varphi) \quad | \quad (\varphi \rightarrow \varphi) \quad | \quad (\varphi \leftrightarrow \varphi) \quad | \quad (\varphi \oplus \varphi)
\end{array}$$

where a is a rational number, e is a natural number greater one, b is a *Boolean variable* and the *arithmetic variable* x is an inherently existential quantified and either real- or integer-valued. We call p a *polynomial* and use a multivariate polynomial with rationals as coefficients to represent it. The **uninterpreted* function f is of a certain *order* $o(f)$ and each of its $o(f)$ arguments are either an arithmetic variable or an *uninterpreted variable* u , which is also inherently existential quantified, but has no domain specified. Than an *uninterpreted equation* e has either an uninterpreted function, an uninterpreted variable or an arithmetic variable as left-hand respectively right-hand side. A *constraint* c compares a polynomial to zero, using a *relation symbol*. Furthermore, we keep constraints in a normalized representation to be able to differ them better.

0.6.2.3 Boolean combinations of constraints and Boolean variables

For more information, check out the docs of [CARL](#).

A formula is stored as a directed acyclic graph, where the intermediate nodes represent the Boolean operations on the sub-formulas represented by the successors of this node. The leaves (nodes without successor) contain either a Boolean variable, a constraint or an uninterpreted equality. Equal formulas, that is formulas being leaves and containing the same element or formulas representing the same operation on the same sub-formulas, are stored only once.

The construction of formulas, which are represented by the `FormulaT` class, is mainly based on the presented abstract grammar. A formula being a leaf wraps the corresponding objects representing a Boolean variable, a constraint or an uninterpreted equality. A Boolean combination of Boolean variables, constraints and uninterpreted equalities consists of a Boolean operator and the sub-formulas it interconnects. For this purpose we either firstly create a set of formulas containing all sub-formulas and then construct the `Formula` or (if the formula shall not have more than three sub-formulas) construct the formula directly passing the operator and sub-formulas. Formulas, constraints and uninterpreted equalities are non-mutable, once they are constructed.

We give a small example constructing the formula

$$(\neg b \wedge x^2 - y < 0 \wedge 4x + y - 8y^7 = 0) \rightarrow (\neg(x^2 - y < 0) \vee b),$$

with the Boolean variable `b` and the real-valued variables `x` and `y`, for demonstration. Furthermore, we construct the UF formula

$$v = f(u, u) \oplus w \neq u$$

with `u`, `v` and `w` being uninterpreted variables of not specified domains `S` and `T`, respectively, and `f` is an uninterpreted function with not specified domain $T^{S \times S}$.

Firstly, we show how to create real valued (integer valued analogously with `VT_INT`), Boolean and uninterpreted variables:

```
carl::Variable x = smtrat::newVariable( "x", carl::VariableType::VT_REAL );
carl::Variable y = smtrat::newVariable( "y", carl::VariableType::VT_REAL );
carl::Variable b = smtrat::newVariable( "b", carl::VariableType::VT_BOOL );
carl::Variable u = smtrat::newVariable( "u", carl::VariableType::VT_UNINTERPRETED );
carl::Variable v = smtrat::newVariable( "v", carl::VariableType::VT_UNINTERPRETED );
carl::Variable w = smtrat::newVariable( "w", carl::VariableType::VT_UNINTERPRETED );
```

Uninterpreted variables, functions and function instances combined in equations or inequalities comparing them are constructed the following way.

```
carl::Sort sortS = smtrat::newSort( "S" );
carl::Sort sortT = smtrat::newSort( "T" );
carl::UVariable uu( u, sortS );
carl::UVariable uv( v, sortT );
carl::UVariable uw( w, sortS );
carl::UninterpretedFunction f = smtrat::newUF( "f", sortS, sortS, sortT );
carl::UFIInstance f1 = smtrat::newUFIInstance( f, uu, uw );
carl::UEquality ueqA( uv, f1, false );
carl::UEquality ueqB( uw, uu, true );
```

Next we see an example how to create polynomials, which form the left-hand sides of the constraints:

```
smtrat::Poly px( x );
smtrat::Poly py( y );
smtrat::Poly lhsA = px.pow(2) - py;
smtrat::Poly lhsB = smtrat::Rational(4) * px + py - smtrat::Rational(8) * py.pow(7);
```

Constraints can then be constructed as follows:

```
smtrat::ConstraintT constraintA( lhsA, carl::Relation::LESS );
smtrat::ConstraintT constraintB( lhsB, carl::Relation::EQ );
```

Now, we can construct the atoms of the Boolean formula

```
smtrat::FormulaT atomA( constraintA );
smtrat::FormulaT atomB( constraintB );
smtrat::FormulaT atomC( b );
smtrat::FormulaT atomD( ueqA );
smtrat::FormulaT atomE( ueqB );
```

and the formulas itself (either with a set of arguments or directly):

```
smtrat::FormulasT subformulasA;
subformulasA.insert( smtrat::FormulaT( carl::FormulaType::NOT, atomC ) );
subformulasA.insert( atomA );
subformulasA.insert( atomB );
smtrat::FormulaT phiA( carl::FormulaType::AND, std::move(subformulasA) );
smtrat::FormulaT phiB( carl::FormulaType::NOT, atomA );
smtrat::FormulaT phiC( carl::FormulaType::OR, phiB, atomC );
smtrat::FormulaT phiD( carl::FormulaType::IMPLIES, phiA, phiC );
smtrat::FormulaT phiE( carl::FormulaType::XOR, atomD, atomE );
```

Note, that \wedge and \vee are n -ary constructors, \neg is a unary constructor and all the other Boolean operators are binary.

0.6.2.4 Normalized constraints

A normalized constraint has the form

$$a_1 \overbrace{x_{1,1}^{e_{1,1}} \cdot \dots \cdot x_{1,k_1}^{e_{1,k_1}}}^{m_1} + \dots + a_n \overbrace{x_{n,1}^{e_{n,1}} \cdot \dots \cdot x_{n,k_n}^{e_{n,k_n}}}^{m_n} + d \sim 0$$

with $n \geq 0$, the i th coefficient a_i being an integral number ($\neq 0$), d being a integral number, x_{i,j_i} being a real- or integer-valued variable and e_{i,j_i} being a natural number greater zero (for all $1 \leq i \leq n$ and $1 \leq j_i \leq k_i$). Furthermore, it holds that $x_{i,j_i} \neq x_{i,l_i}$ if $j_i \neq l_i$ (for all $1 \leq i \leq n$ and $1 \leq j_i, l_i \leq k_i$) and $m_{i_1} \neq m_{i_2}$ if $i_1 \neq i_2$ (for all $1 \leq i_1, i_2 \leq n$). If n is 0 then d is 0 and \sim is either $=$ or $<$. In the former case we have the normalized representation of any variable-free consistent constraint, which semantically equals `true`, and in the latter case we have the normalized representation of any variable-free inconsistent constraint, which semantically equals `false`. Note that the monomials and the variables in them are ordered according the `\polynomialOrder` of `\carl`. Moreover, the first coefficient of a normalized constraint (with respect to this order) is always positive and the greatest common divisor of a_1, \dots, a_n , d is 1. If all variable are integer valued the constraint is further simplified to

$$\frac{a_1}{g} \cdot m_1 + \dots + \frac{a_n}{g} \cdot m_n + d' \sim '0,$$

where g is the greatest common divisor of a_1, \dots, a_n ,

$$\sim ' = \begin{cases} \leq, & \text{if } \sim \text{ is } < \\ \geq, & \text{if } \sim \text{ is } > \\ \sim, & \text{otherwise} \end{cases}$$

and

$$d' = \begin{cases} \lceil \frac{d}{g} \rceil & \text{if } \sim \text{ is } \leq \\ \lfloor \frac{d}{g} \rfloor & \text{if } \sim \text{ is } \geq \\ \frac{d}{g} & \text{otherwise} \end{cases}$$

If additionally $\frac{d}{g}$ is not integral and $\sim '$ is $=$, the constraint is simplified $0 < 0$, or if $\sim '$ is \neq , the constraint is simplified $0 = 0$.

We do some further simplifications, such as the elimination of multiple roots of the left-hand sides in equations and inequalities with the relation symbol \neq , e.g., $x^3 = 0$ is simplified to $x = 0$. We also simplify constraints whose left-hand sides are obviously positive (semi)/negative (semi) definite, e.g., $x^2 \leq 0$ is simplified to $x^2 = 0$, which again can be simplified to $x = 0$ according to the first simplification rule.

0.6.2.5 Linking

[Todo: example how linking works]

0.7 Other tools

0.7.1 Benchmarking

Alongside SMT-RAT, `benchmax` allows to easily run solvers on benchmarks and export the results. See [Benchmax](#) for more information.

0.7.2 Delta debugging

Delta debugging describes a generic debugging approach based on automated testing. Given a program and an input that provokes a certain behavior (for example an error) delta debugging is the process of iteratively changing the input, retaining the specific behavior. Usually, as the ultimate goal is a minimal example that triggers some bug by the application of a set of transformation rules.

0.7.2.1 SMT-RAT's own delta tool

SMT-RAT has its own tool for delta debugging, see [Delta](#) for more information.

0.7.2.2 Using ddSMT

Currently, we recommend using ddSMT as the state-of-the-art tool for delta debugging.

For a complete guide, we refer to the [documentation of ddSMT](#).

Here, we give instructions for the most common use cases when developing with SMT-RAT. There are mainly two common scenarios where we use delta debugging: If there is a failing assertion or if SMT-RAT returns a wrong result. For these scenarios, we provide wrapper scripts preparing the input to ddsmt properly.

First, install ddSMT and z3:

```
pip3 install ddsmt  
sudo apt install z3
```

In case of a *failing assertion* on `in_file.smt2`, run

```
~/src/smtrat/utilities/ddsmt/ddsmt-assertion in_file.smt2 out_file.smt2 path_to_solver
```

In case SMT-RAT returns a *wrong result* on `in_file.smt2`, run

```
~/src/smtrat/utilities/ddsmt/ddsmt-wrong in_file.smt2 out_file.smt2 path_to_solver
```

0.7.3 Analyzer

The `smtrat-analyzer` library can analyze static properties of input formulas (such as number of variables, degrees, CAD projections, ...).

To use it from the CLI, build smtrat using `CLI_ENABLE_ANALYZER=ON` and `SMTRAT_DEVOPTION_Statistics=ON`. A single input file can be analyzed by running `./smtrat --analyze.enabled --stats.print input.smt2`; properties will be printed as `statistics` object, note that the solver will not be called. For further options, see `./smtrat --help`.

To collect properties of all formulas of a benchmark sets, the analyzer can be used with `benchmax`. To do so, specify add the binary as an `smtrat-analyzer` solver; for more information, see the `benchmax` subpage.

0.7.4 Preprocessing

WIP

0.7.5 Benchmax

Benchmax is a tool for automated benchmarking. Though it was developed and is primarily used for testing SMT solvers, it aims to be agnostic of the tools and formats as good as possible.

Its fundamental model is to load a list of tools and a list of benchmarks, run every tool with every benchmark, obtain the results and output all the results. While the benchmarks are fixed to a single file, we allow to choose between different [tool interfaces](#), [execution backends](#) and output formats.

0.7.5.1 General usage

Benchmax could be called as follows:

```
./benchmax -T 1m -M 4Gi -S /path/to/smtrat-static -X out.xml -b ssh --ssh.node user@127.0.0.1@5\#9 -D /path/to/benchmark/set
```

This will run benchmax with the SMT-RAT tool at `/path/to/smtrat-static` on the `/path/to/benchmark/set` with a time limit of 1 minute and a memory limit of 4 gigabyte. The result will be written to `out.xml`. The execution is done on an SSH backend on localhost with 9 connections and executing 5 parallel jobs per connection.

It is recommended to use static builds to avoid issues with missing libraries.

For more information, run `./benchmax --help`.

0.7.5.1.1 Collecting statistics Some tools like SMT-RAT or Z3 can provide statistics about the solving process for each individual benchmark. By using the `-s` respectively `--statistics`, statistics are collected and stored in the output file.

0.7.5.1.1.1 Large output It is recommended to aggregate statistics as much as possible inside SMT-RAT. However, if the output might get large, you might want to use `--use-tmp` to prevent benchmax running out of memory.

0.7.5.1.2 Working with the results In the SMT-RAT repository, two utilities for converting the result XML file are included:

- `utilities/benchmax/xml2ods.py` for converting it to a *Flat XML LibreOffice Calc Sheet*.
- An XML filter `utilities/benchmax/OOCImporter.xsl` for converting it to a *Flat XML LibreOffice Calc Sheet*.
- `utilities/benchmax/evaluation` is a small python library for importing the results into Python (or a Jupyter notebook), inspecting the results and preparing plots. For more information, see [Benchmax python utility](#).

0.7.5.2 Tools

A tool represents a binary that can be executed on some input files. It is responsible for deciding whether it can be applied to some given file, building a command line to execute it and retrieve additional results from `stdout` or `stderr`.

The generic tool class `benchmax::Tool` can be used as a default. It will run the given binary on any input file and benchmax records the exit code as well as the runtime.

If more information is needed a new tool class can be derived that overrides or extends the default behaviour. Some premade tools are available (and new ones should be easy to create):

- [benchmax::MathSAT](#)
- [benchmax::Minisatp](#)
- [benchmax::SMTRAT](#)
- [benchmax::SMTRAT_OPB](#)
- [benchmax::Z3](#)

0.7.5.3 Backends

A **backend** offers the means to run the tasks in a specific manner. A number of different backends are implemented that allow for using benchmax in various scenarios.

0.7.5.3.1 LocalBackend The **LocalBackend** can be used to execute benchmarks on a local machine easily. It does not allow for any parallelization but simply executes all tasks sequentially.

0.7.5.3.2 SlurmBackend The **SlurmBackend** employs the Slurm workload manager to run benchmarks on a cluster. It essentially collects all tasks that shall be run in a file and creates an appropriate slurm submit file. It then submits this job, waits for it to finish and collects the results from the output files.

The Slurm backend supports starting (`--mode execute`) and collecting results (`--mode collect`) separately, thus avoiding the need for running benchmax in a screen.

0.7.5.3.3 SSHBackend The **SSHBackend** can be used if you have multiple workers that can be reached via ssh (essentially a cluster without a batch job system). It allows to configure one or more computing nodes and manually schedules all the jobs on the different computing nodes.

Using `--ssh.node`, a node is specified in the format `<user>[:<password>]@<hostname>[:<port = 22>] [<cores = 1>][#<connections = 1>]`; benchmax will then connect to the specified server with given credentials `connections` times and will run `cores` threads per connection. Thus, the number of overall parallel threads equals `connections * cores` (note that enough memory should be available on the node).

By setting the flag `--ssh.resolvedeps`, dependencies (i.e. dynamic libraries) are resolved and uploaded with the solver.

0.7.5.3.4 CondorBackend (unstable) The **CondorBackend** is aimed at the HTCondor batch system and works similar to the SlurmBackend. Note however that this backend is not well tested and we therefore consider it unstable.

0.7.5.4 Troubleshooting

- If benchmax terminates due to a segmentation fault without any message, then it probably exceeded available memory. A common error is that SMT-RAT has produced too much logging output. To fix this, simply build SMT-RAT with `LOGGING=OFF`.

0.7.5.5 Benchmax python utility

This tool allows loading XML files from Benchmax into a pandas dataframe, inspecting the results and visualizing them.

This is useful for working with the results in a Jupyter notebook.

Dependencies:

- `pandas`
- `matplotlib`
- `pillow`
- `numpy`
- `tikzplotlib`

```
pip3 install pandas matplotlib tikzplotlib numpy pillow
```

0.7.5.5.1 Loading XMLs into a pandas dataframe

First, install this directory as python library; e.g. on Ubuntu
cd ~/.local/lib/python3.8/site-packages/ # path to your python site-packages directory
ln -s ~/src/smtrat/utilities/benchmax/ # path to the benchmax utility

In your Jupyter notebook:

```
import benchmax.evaluation as ev
df = ev.xml_to_pandas("path_to/stats_file.xml", {"smtrat-static": "solver_name"}, ["statistics_name_1", "statistics_name_2"]) # second and third parameter is optional
```

or, to load multiple XMLs into one:

```
import benchmax.evaluation as ev
df = ev.xls_to_pandas({"path_to/stats_file_1.xml": {"smtrat-static": "solver_name_1"}, "path_to/stats_file_2.xml": {"smtrat-static": "solver_name_2"}}, ["statistics_name_1", "statistics_name_2"]) # second parameter is optional
```

This will create a dataframe with columns (solver_name_1, "answer"), (solver_name_1, "runtime"), (solver_name_1, "statistics_name_1") etc (as multi-index).

0.7.5.5.2 Computing a virtual best solver

A virtual best is a solver that behaves optimal on each input w.r.t. a set of solvers. It is computed by selecting for each benchmark instance the solver with shortest running time.

To compute the virtual best solver named VB w.r.t. solver1,solver2 and solver3 and considering statistics statistics_name_1.

```
df = df.join(ev.virtual_best(df, ["solver1", "solver2", "solver3"], "VB", ['statistics_name_1']))
```

0.7.5.5.3 Plotting

We provide the performance_profile and scatter_plot functions to easily generate a performance profiles and scatter plots.

0.7.5.5.4 Show a summary of an XML file

There are several methods for quickly inspecting the results of a run provided. For instance, the following script can be used to view a summary of a singel XML file and to show instances with wrong results or segmentation faults:

```
#!/usr/bin/env python3
import benchmax.evaluation as ev
import sys
df = ev.xml_to_pandas(sys.argv[1])
ev.inspect(df)
```

SMT-RAT provides a small utility script for showing a summary of one or more XML files:

```
~/Code/smtrat/utilities/benchmax/view.py result_1.xml result_2.xml
```

0.7.5.5.5 Exporting data to Latex

0.7.5.5.5.1 Pandas dataframe to latex table

```
df.to_latex(buf='file.tex')
```

0.7.5.5.5.2 matplotlib to tikz plot

see [tikzplotlib](#)

0.7.6 Delta

0.7.6.1 Delta debugging

Delta debugging describes a generic debugging approach based on automated testing. Given a program and an input that provokes a certain behavior (for example an error) delta debugging is the process of iteratively changing the input, retaining the specific behavior. Each small change to the input represents a **delta** and is the result of some transformation rule. Whenever a change was successful, it is stored and the process continues from this intermediate result. Eventually, there is no transformation left, such that the faulty behavior is retained and the debugging process terminates.

This approach only works, if the transformation rules can neither be chained to form a loop, nor continue infinitely. Usually, as the ultimate goal is a minimal example that triggers some bug, all transformation rules are designed to make the input smaller, in one way or another.

This approach has proven to be very valuable in the context of SAT and SMT solving. However, existing delta debugging tools ? needed a preprocessed input and manual restarts to achieve a fix-point, hence, we decided to include our own delta debugging tool, `delta`, in SMT-RAT. It can be used completely independent of SMT-RAT and is built to be as generic as possible, but focuses on programs operating on SMT-LIB files. It has some knowledge of the semantics of the corresponding logics, but only operates on nodes. Any SMT-LIB construct, that is either a constant or a braced expression, is a node.

The actual transformation rules are implemented in `operations.h` and are enabled in the constructor of the `Producer` class. The implemented rules are rather simple: removing a node, replacing a node by a child node, simplifying a number, replacing a symbol by a constant or eliminating a let expression. These transformations are designed such that they can be extended easily. For a given input `delta` applies each transformation to each node. Each application may produce arbitrarily many *candidate inputs* which are then tested. The first candidate that provokes the error is then adopted, the other candidates are rejected.

When analyzing the behavior, `delta` relies on the exit code of the program. It will run the program on the original input and obtain the original exit code. Whenever the program returns the same exit code, `delta` assumes that the program behaved the same.

0.7.6.1.1 Using delta `delta` is currently contained in the SMT-RAT repository and can be built using `make delta`.

Using `delta` is rather easy. It accepts the input file and the solver as its two main arguments: `./delta -i input.smt2 -s ./solver`. There are a couple of other arguments that are documented in the help: `./delta --help`.

0.7.6.1.1.1 Exit code As `delta` relies on the exit code of the program, make sure that this event results in a unique exit code:

- **Specific assertions:** Return a unique exit code by replacing the assertion with `if (!assertion_< condition) { std::exit(70); }`.
- **Faulty output:** Remove the `(set-info :status unsat)/(set-info :status sat)` line from the benchmark and use the script `utilities/result-incorrect.py` as solver. Note that you need to install z3 first.

0.8 Developers information

- [Code style](#)
- [Documentation](#)
- [Settings](#)
- [Logging](#)
- [Statistics and timing](#)
- [Testing](#)
- [Validation](#)
- [Checkpoints](#)
- [Finding and Reporting Bugs](#)
- [Tools](#)

0.8.1 Code style

Please follow the code style rules from CArL documentation.

For SMT-RAT, we follow a different rule for naming files and namespaces:

- Every file should represent a *module*.
- Every (sub-)folder has its own namespace.

0.8.2 Documentation

For SMT-RAT, there are two sources of documentation: the in-source API documentation and the more manual-like documentation (you are reading right now). While the in-source API documentation is generated based on the doxygen comments from the actual source files, the manual is generated from the markdown files in `doc/markdown/`.

0.8.2.0.1 Writing documentation Note that some of the documentation may be incomplete or rendered incorrectly, especially if you use an old version of doxygen. Here is a list of known problems:

- Comments in code blocks (see below) may not work correctly (e.g. with doxygen 1.8.1.2). See [here](#) for a workaround. This will however look ugly for newer doxygen versions, hence we do not use it.
- Files with `static_assert` statements will be incomplete. A [patch](#) is pending and will hopefully make it into doxygen 1.8.9.
- Member groups (usually used to group operators) may or may not work. There still seem to be a few cases where doxygen [messes up](#).
- Documenting unnamed parameters is not possible. A corresponding [ticket](#) exists for several years.

0.8.2.0.2 Literature references Literature references should be provided when appropriate by citing references from the bibtex database located at `doc/literature.bib` using the `@cite` command. The labels are the last name of the first author and the four-digit year. In case of duplicates, we append lowercase letters.

These references can be used with `@cite` label, for example like this:

```
/***
 * Checks whether the polynomial is unit normal
 * @see @cite GCL92, page 39
 * @return If polynomial is normal.
 */
bool is_normal() const;
```

0.8.2.1 Code comments

0.8.2.1.1 File headers

```
/***
 * @file <filename>
 * @ingroup <groupid1>
 * @ingroup <groupid2>
 * @author <author1>
 * @author <author2>
 *
 * [ Short description ]
 */
```

Descriptions may be omitted when the file contains a single class, either implementation or declaration.

0.8.2.1.2 Namespaces Namespaces are documented in a separate file, found at '`/doc/markdown/codedocs/namespaces.dox`'

0.8.2.1.3 Class headers

```
/***
 * @ingroup <groupid>
 * [ Description ]
 * @see <reference>
 * @see <OtherClass>
 */
```

0.8.2.1.4 Method headers

```
/***
 * [ Usage Description ]
 * @param <p1> [ Short description for first parameter ]
 * @param <p2> [ Short description for second parameter ]
 * @return [ Short description of return value ]
 * @see <reference>
 * @see <otherMethod>
 */
```

These method headers are written directly above the method declaration. Comments about the implementation are written above the or inside the implementation.

The `see` command is used likewise as for classes.

0.8.2.1.5 Method groups There are some cases when documenting each method is tedious and meaningless, for example operators. In this case, we use doxygen method groups.

For member operators (for example operator`+=`), this works as follows:

```
/// @name In-place addition operators
/// @{
/***
 * Add something to this polynomial and return the changed polynomial.
 * @param rhs Right hand side.
 * @return Changed polynomial.
 */
MultivariatePolynomial& operator+=(const MultivariatePolynomial& rhs);
MultivariatePolynomial& operator+=(const Term<Coeff>& rhs);
MultivariatePolynomial& operator+=(const Monomial& rhs);
MultivariatePolynomial& operator+=(Variable::Arg rhs);
MultivariatePolynomial& operator+=(const Coeff& rhs);
/// @}
```

0.8.2.2 Writing out-of-source documentation

Documentation not directly related to the source code is written in Markdown format, and is located in `/doc/markdown/`.

0.8.3 Settings

As described in sec::modules, each module has a settings template parameter.

0.8.3.0.1 Dynamic settings These settings are considered at compile time. For certain purposes, it is unhandy to recompile SMT-RAT for every parameter. For this, module parameters can be set

- via the command line using `--module.parameter key1=value1 --module.parameter key2=value2 ...` or
- via the `set-option` command of SMT-LIB: (`set-option :key "value"`) (note that you need to pass every value as a string, even if it represents a number!).

These parameters can be accessed (most appropriately in a settings object of a module) via `int my_setting = settings_module().get("my_module.my_setting", 2)` or `std::string my_setting = settings_module().get("my_module.my_setting", "default_value")`. Note that the second parameter is the default value also specifying the type of the setting to which it is parsed.

0.8.4 Logging

0.8.4.0.1 Logging frontend The frontend for logging is defined in `logging.h`.

It provides the following macros for logging:

- `SMTRAT_LOG_TRACE(channel, msg)`
- `SMTRAT_LOG_DEBUG(channel, msg)`
- `SMTRAT_LOG_INFO(channel, msg)`

- SMTRAT_LOG_WARN(channel, msg)
- SMTRAT_LOG_ERROR(channel, msg)
- SMTRAT_LOG_FATAL(channel, msg)
- SMTRAT_LOG_FUNC(channel, args)
- SMTRAT_LOG_FUNC(channel, args, msg)
- SMTRAT_LOG_ASSERT(channel, condition, msg)
- SMTRAT_LOG_NOTIMPLEMENTED()
- SMTRAT_LOG_INEFFICIENT()

Where the arguments mean the following:

- **channel**: A string describing the context. For example "smtrat.parser".
- **msg**: The actual message as an expression that can be sent to a std::stringstream. For example "foo: " << foo.
- **args**: A description of the function arguments as an expression like msg.
- **condition**: A boolean expression that can be passed to assert().

Typically, logging looks like this:

```
bool checkStuff(Object o, bool flag) {
    SMTRAT_LOG_FUNC("smtrat", o << ", " << flag);
    bool result = o.property(flag);
    SMTRAT_LOG_TRACE("smtrat", "Result: " << result);
    return result;
}
```

Logging is enabled (or disabled) by the LOGGING macro in CMake. The log levels for each channel can be configured in the configure_logging() method in cli/smtratSolver.cpp.

0.8.5 Statistics and timing

0.8.5.0.1 Statistics

SMT-RAT has the ability to collect statistics after the solving process is finished.

For enabling this feature, the SMTRAT_DEVOPTION_Statistics needs to be turned on in the CMake settings.

0.8.5.0.1.1 Collecting statistics

A statistics object can be created by inheriting from `Statistics.h`:

```
#pragma once
#include <smtrat-common/statistics/Statistics.h>
#ifndef SMTRAT_DEVOPTION_Statistics
namespace smtrat {
namespace myModule {
class MyStatistics : public Statistics {
private:
    std::size_t mCounter = 0;
public:
    void collect() { // called after the solving process to collect statistics
        Statistics::addKeyValuePair("counter", mCounter);
    }
    void count() { // user defined
        ++mCounter;
    }
};
}
#endif
```

Note that neither the key nor the value are allowed to contain spaces, (or).

This is then instantiated by calling

```
#ifdef SMTRAT_DEVOPTION_Statistics
auto& myStatistics = statistics_get<myModule::MyStatistics>("MyModuleName");
#endif
```

or, as shortcut

```
SMTRAT_STATISTICS_INIT(myModule::MyStatistics, myStatistics, "MyModuleName")
```

and statistics can be collected by calling the user defined operations (e.g. `myStatistics.count()`).

All statistics-related code should be encapsulated by the `SMTRAT_DEVOPTION_Statistics` flag. Alternatively, code can be encapsulated in `SMTRAT_STATISTICS_CALL()`.

0.8.5.0.2 Timing

The statistics framework has the ability to easily collect timings.

The following code will measure the total running time of the code block as well as the number of times the code block was executed:

```
class MyStatistics : public Statistics {
private:
    carl::statistics::timer mTimer;

public:
    void collect() {
        Statistics::addKeyValuePair("timer", mTimer);
    }
    auto& timer() {
        return mTimer;
    }
};

SMTRAT_STATISTICS_INIT(myModule::MyStatistics, myStatistics, "MyModuleName")

SMTRAT_TIME_START(start);
// expensive code
SMTRAT_TIME_FINISH(myStatistics.timer(), start);
```

The measured timings will then appear alongside the statistics.

Note that the resolution of these measurements is in 1 millisecond. Thus, be careful when interpreting results; especially, it measured code blocks should be big enough.

0.8.6 Testing

SMT-RAT and CArL use [GoogleTest](#) for unit testing. To do so, go to the `src/tests/` folder and create a test analogously to the other tests (i.e. creating a new folder, adapting the `CMakeList.txt` in the new folder and in the `tests` folder). *Note that in the SMT-RAT repository are some obsolete tests.*

0.8.6.0.1 Utilities For creating CArL data structure in unit tests, we refer to `carl-io/parser/Parser.h`.

0.8.7 Validation

For debugging purposes, it can be useful to verify intermediate results (explanations, lemmas, etc) using an external SMT solver. SMT-RAT allows to store formulas during the solving process which are written to a `smt2` file once the solving process is finished.

0.8.7.0.1 Enabling this feature For enabling this feature, the `SMTRAT_DEVOPTION_Validation` needs to be turned on in the CMake settings.

0.8.7.0.2 Logging formulas The API allows to create a *validation point* with a given channel and a name. The channel and name should identify the validation point uniquely. The channel (e.g. `smtrat.modules.vs`) can be used to turn on and off validation points (similarly to logging channels) while the name (e.g. `substitution->result`) further distinguishes validation points within a channel.

To initialize a validation point with channel and name and store its reference to a variable, use

```
SMTRAT_VALIDATION_INIT(channel, variable);
```

Hint: to put it in a static variable, use

```
SMTRAT_VALIDATION_INIT_STATIC(channel, variable);
```

To an initialized validation point stored in a variable, we can add a formula to be assumed to be satisfiable (consistent = true) or unsatisfiable (consistent = false). Each formula added to a validation point gets a unique index (given incrementally), which is also logged in the given channel with debug level.

```
SMTRAT_VALIDATION_ADD_TO(variable, name, formula, consistent);
```

To combine the two steps above, use:

```
SMTRAT_VALIDATION_ADD(channel, name, formula, consistent);
```

0.8.7.0.3 Command line usage By appending the command line parameter `--validation.export-smtlib`, the formulas are stored to an smtlib file (by default `validation.smt2`, can be customized using `--validation.smtlib-filename`).

Note that all channels of interest need to be activated explicitly with `--validation.channel channel1 --validation.channel channel2 ...`. Furthermore, `--validation.channel path.to` will activate all channels starting with `path.to`, i.e. `path.to.channel1, path.to.channel2` etc.

0.8.7.0.4 Segmentation faults Note that the formulas are only written to a file if smtrat terminates without an segmentation fault. If there is an assertion failing, set DEVELOPER=OFF in cmake.

0.8.8 Checkpoints

Checkpoints are useful for debugging a run of SMT-RAT, i.e. if an implementation of SMT-RAT behaves the same as some other implementation or some pseudocode.

To do so, we can insert checkpoints with a channel, a name, and a list of arguments of arbitrary type into the code. During a run, a sequence of checkpoints is produced which represent a run of the algorithm. We then can specify test cases which set up a reference run and then start the solver on an according input. SMT-RAT will then check during this run whether all checkpoints are hit in the correct order and will give output on that.

0.8.8.0.1 Enabling this feature You need to turn on the CMake option `SMTRAT_DEVOPTION_Checkpoints`.

0.8.8.0.2 Specyfing checkpoints Insert

```
SMTRAT_CHECKPOINT("channel_name", "checkpoint_name", parameter1, parameter2, ...);
```

into the code.

0.8.8.0.3 Setting up a reference run Create a test case in `src/test`. Include `#include <smtrat-common/smtrat-common.h>` and add checkpoints using

```
SMTRAT_ADD_CHECKPOINT("channel_name", "checkpoint_name", false, parameter1, parameter2, ...)
```

or

```
SMTRAT_ADD_CHECKPOINT("channel_name", "checkpoint_name", true, parameter1, parameter2, ...)
```

to assert that a checkpoint is reached.

Afterwards, run the respective SMT-RAT code.

You can clear all checkpoints of a channel using

```
SMTRAT_CLEAR_CHECKPOINT("channel_name");
```

to reset a channel, that is, removing all checkpoints and resetting the pointer.

0.8.9 Finding and Reporting Bugs

This page is meant as a guide for the case that you find a bug or any unexpected behaviour. We consider any of the following events a (potential) bug:

- SMT-RAT crashes.
- A library used through SMT-RAT crashes.
- SMT-RAT gives incorrect results.
- SMT-RAT does not terminate (for reasonably sized inputs).
- SMT-RAT does not provide a method or functionality that should be available according to this documentation.
- SMT-RAT does not provide a method or functionality that you consider crucial or trivial for some of the data-structures.
- Compiling SMT-RAT fails.
- Compiling your code using SMT-RAT fails and you are pretty sure that you use SMT-RAT according to this documentation.

In any of the above cases, make sure that:

- You have installed all necessary dependencies in the required versions.
- You work on something that is similar to a system listed as supported platform at [getting_started](#).
- You can (somewhat reliably) reproduce the error with a (somewhat) clean build of SMT-RAT. (i.e., you did not screw up the CMake flags, see [build_cmake](#) for more information)
- You compile either with `CMAKE_BUILD_TYPE=DEBUG` or `DEVELOPER=ON`. This will give additional warnings during compilation and enable assertions during runtime. This will slow down SMT-RAT significantly, but detect errors before an actual crash happens and give a meaningful error message in many cases.

If you are unable to solve issue yourself or you find the issue to be an actual bug in SMT-RAT, please do not hesitate to contact us. You can either contact us via email (if you suspect a configuration or usage issue on your side) or create a ticket in our bug tracker (if you suspect an error that is to be fixed by us). The bug tracker is available at <https://github.com/thc-rwth/smtrat/issues>.

When sending us a mail or creating a ticket, please provide us with:

- Your system specifications, including versions of compilers and libraries listed in the dependencies.
- The SMT-RAT version (release version or git commit id).
- A minimal working example.
- A description of what you would expect to happen.
- A description of what actually happens.

0.9 README

This is dead code.

The expressions were meant as an alternative to `carl::Formula`. Instead of having own theory constraints for each theory, here, everything is stored in a single tree structure. This would make theory combinations easier (think of a variable being interpreted both as uninterpreted and as arithmetic variable); furthermore, handling `ite` expression is easier: currently, we need to resolve `ite` expressions within the theory in the parser, leading to bad performance on some examples.

0.10 NewCoveringModule #and

Author: Philip Kroll (Philip.Kroll@rwth-aachen.de)

0.10.1 Introduction

New implementation of the CDCAC algorithm, also called the covering algorithm. The basic implementation is based on the paper [Deciding the Consistency of Non-Linear Real Arithmetic Constraints with a Conflict Driven Search Using Cylindrical Algebraic Coverings](#). In short : The algorithm is a variant of Cylindrical Algebraic Decomposition (CAD) adapted for satisfiability, where solution candidates (sample points) are constructed incrementally, either until a satisfying sample is found or sufficient samples have been sampled to conclude unsatisfiability. The choice of samples is guided by the input constraints and previous conflicts.

0.10.2 Usage

In this implementation as much code as possible from src/smtrat-cadcells. This also includes the projection operators implemented in src/smtrat-cadcells/operators and the representation heuristics implemented in src/smtrat-cadcells/representation. We also reuse the code from [src/smtrat-mcsat/variableordering/VariableOrdering.h](#) to calculate the used variable ordering. Change of the strategy to calculate the variable ordering, the used heuristics or the used projection operator, can be done by changing the respective settings in src/smtrat-modules/NewCoveringModule/NewCoveringSettings.h

0.10.3 Efficiency

The implementation also supports backtracking and incrementality, both of which are not covered in detail by the paper. The following cases are possible depending on what the previous result of the Covering Algorithm was :

- Previous result was SAT, i.e. a satisfying assignment was found and the cached coverings for all dimensions are partial
 - addConstraintSAT() is called with the new constraint and these are checked for satisfiability. The lowest level with an unsatisfied new constraints is returned. This also means that when all constraints SAT is concluded and no calculations are done.
 - removeConstraintSAT() is called with constraints that have to be removed from the cached partial coverings for all dimensions. This only makes the covering smaller, and the assignment is still satisfying.
- Previous result was UNSAT, i.e. the stored covering for level 0 is complete and thus no satisfying assignment could be found.
 - addConstraintUNSAT() is called with the new constraint. As the covering is unsatisfiable anyways these constraints are just added without further calculations.
 - removeConstraintUNSAT() is called with constraints that have to be removed from the cached complete coverings all dimensions.

0.11 Todo List

Global `smtrat::parser::QualifiedIdentifierParser::checkQualification` (`const Identifier &identifier, const carl::Sort &`) `const`

Check what can be checked here.

Global `smtrat::parser::Theories::defineFunction` (`const std::string &name, const std::vector< types::VariableType > &arguments, const carl::Sort &sort, const types::TermType &definition`)

check that definition matches the sort

0.12 Hierarchical Index

0.12.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

smrat::AbstractModuleFactory	235
smrat::ModuleFactory< Module >	1238
smrat::parser::AbstractTheory	235
smrat::parser::ArithmeticTheory	244
smrat::parser::BitvectorTheory	313
smrat::parser::BooleanEncodingTheory	320
smrat::parser::CoreTheory	488
smrat::parser::UninterpretedTheory	1859
smrat::cad::sample_compare::absvalue	238
std::allocator< T >	
smrat::fixedsize_allocator< void >	753
smrat::analyzer::AnalysisSettings	238
smrat::AnnotatedBVTerm	239
smrat::cadcells::representation::approximation::ApxSettings	241
benchmax::slurm::ArchiveProperties	243
smrat::parser::types::ArithmeticTheory	246
smrat::execution::Assertion	247
smrat::cad::preprocessor::AssignmentCollector	247
smrat::mcsat::arithmetic::AssignmentFinder	248
smrat::mcsat::smtaf::AssignmentFinder< Settings >	249
smrat::mcsat::arithmetic::AssignmentFinder_ctx	249
smrat::mcsat::arithmetic::AssignmentFinder_detail	250
smrat::mcsat::smtaf::AssignmentFinder_SMT	251
smrat::cadcells::datastructures::detail::AssignmentProperties	251
smrat::parser::Attribute	252
smrat::AxiomFactory	254
benchmax::Backend	255
benchmax::CondorBackend	468
benchmax::LocalBackend	1078
benchmax::SSHBackend	1680
benchmax::SlurmBackend	1636
smrat::Backend< Settings >	257
smrat::BackendLink	260
smrat::BackendSynchronisation	261
smrat::Backtrackable< UnionFind >	261
smrat::cadcells::datastructures::BaseDerivation< Properties >	263
smrat::cad::BaseProjection< Settings >	265
smrat::cad::ModelBasedProjection< ProjectionSettings >	1199
smrat::cad::Projection< Settings >	1492
smrat::cad::ModelBasedProjection< incrementality, backtracking, Settings >	1199
smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >	1205
smrat::cad::Projection< incrementality, backtracking, Settings >	1492
smrat::cad::Projection< Incrementality::FULL, BT, Settings >	1509
smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >	1503
smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >	1514
smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >	1520
smrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >	1526
smrat::qe::cad::Projection< Settings >	1497
smrat::cad::BaseSettings	270
smrat::qe::cad::CADSettings	386
smrat::cad::Origin::BaseType	273
smrat::mcsat::onecell::BCFilteredSettings	277

smrat::mcsat::onecell::BCFilteredAllSelectiveSettings	275
smrat::mcsat::onecell::BCFilteredAllSettings	276
smrat::mcsat::onecell::BCFilteredBoundsSettings	276
smrat::mcsat::onecell::BCFilteredSamplesSettings	276
smrat::mcsat::onecell::BCSettings	277
smrat::mcsat::onecell::BCApproximationSettings	275
benchmax::BenchmarkResult	305
benchmax::BenchmarkSet	307
benchmax::settings::BenchmarkSettings	308
smrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >	309
smrat::Bimap< Expansion, const carl::Variable, &Expansion::mRationalization, const carl::Variable, &Expansion::mDiscretization >	309
smrat::Bimap< Linearization, const Poly, &Linearization::mNormalization, const Poly, &Linearization::m← Linearization >	309
smrat::expression::BinaryExpression	311
smrat::datastructures::BinaryHeap< T >	312
smrat::parser::types::BitvectorTheory	316
smrat::BlastedConstr	317
smrat::BlastedPoly	318
smrat::mcsat::Bookkeeping	319
smrat::BoolUEQRewriter	324
smrat::cadcells::datastructures::Bound	325
smrat::lra::Bound< T1, T2 >	326
smrat::mcsat::fm::Bound	335
smrat::vb::Bound< T >	336
smrat::Branching	341
smrat::BVAnnotation	342
smrat::BVDirectEncoder	344
smrat::by_address_hasher< IterType >	372
smrat::cad::CAD< Settings >	372
smrat::qe::cad::CAD< Settings >	375
smrat::qe::cad::CAD< smrat::qe::cad::CADSettings >	375
smrat::cad::CADConstraints< BT >	377
smrat::cad::CADConstraints< ProjectionSettings::backtracking >	377
smrat::cad::CADConstraints< Settings::backtracking >	377
smrat::cad::CADCore< CH >	381
smrat::cad::CADCore< CoreHeuristic::BySample >	381
smrat::cad::CADCore< CoreHeuristic::EnumerateAll >	381
smrat::cad::CADCore< CoreHeuristic::Interleave >	382
smrat::cad::CADCore< CoreHeuristic::PreferProjection >	382
smrat::cad::CADCore< CoreHeuristic::PreferSampling >	383
smrat::qe::cad::CADElimination	383
smrat::cad::CADPreprocessor	383
smrat::cad::CADPreprocessorSettings	385
smrat::cadcells::representation::cell< H >	388
smrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL >	388
smrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION >	389
smrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_EW >	389
smrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ >	389
smrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS >	390
smrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ >	390
smrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EW >	390
smrat::cadcells::operators::properties::cell_connected	391
smrat::cadcells::representation::approximation::CellApproximator	391
smrat::cadcells::datastructures::CellRepresentation< P >	392
delta::Checker	393
benchmax::slurm::ChunkedSubmitfileProperties	394
Minisat::Clause	396

smrat::mcsat::ClauseChain	401
smrat::sat::detail::ClauseChecker	403
smrat::CMakeOptionPrinter	404
Minisat::CMap< T >	404
smrat::CollectionWithOrigins< Element, Origin >	462
smrat::CollectionWithOrigins< carl::Variable, FormulaT >	462
smrat::ICPModule< Settings >::comp	464
smrat::cadcells::datastructures::CompoundMax	464
smrat::cadcells::datastructures::CompoundMin	464
smrat::vs::Condition	465
smrat::mcsat::fm::ConflictGenerator< Comparator >	470
smrat::cad::ConflictGraph	470
smrat::parser::Theories::ConstantAdder	472
smrat::cad::CADConstraints< BT >::ConstraintComparator	473
smrat::cad::preprocessor::ConstraintUpdate	473
smrat::ConstrTree	474
delta::Consumer	475
smrat::LRAModule< Settings >::Context	476
smrat::icp::ContractionCandidate	477
smrat::icp::contractionCandidateComp	482
smrat::icp::ContractionCandidateManager	482
smrat::parser::conversion::Converter< To >	484
smrat::parser::conversion::Converter< FormulaT >	484
smrat::parser::conversion::Converter< Poly >	485
smrat::parser::conversion::Converter< Res >	484
smrat::parser::conversion::Converter< types::BVTerm >	485
benchmax::settings::CoreSettings	486
smrat::settings::CoreSettings	487
smrat::parser::types::CoreTheory	490
smrat::cadcells::representation::covering< H >	491
smrat::mcsat::arithmetic::Covering	491
smrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING >	493
smrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING_EW >	493
smrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING >	493
smrat::cadcells::datastructures::CoveringDescription	494
smrat::cadcells::datastructures::CoveringRepresentation< P >	494
smrat::mcsat::onecellcad::recursive::CoverNullification	496
smrat::CycleEnumerator< FHG, Collector >	576
benchmax::Database	576
benchmax::DBAL	578
Minisat::DeepEqual< K >	582
Minisat::DeepHash< K >	582
smrat::mcsat::fm::DefaultComparator	582
smrat::mcsat::fm::DefaultSettings	583
smrat::mcsat::smtaf::DefaultSettings	583
smrat::mcsat::vs::DefaultSettings	583
smrat::cad::projection_compare::degree	584
smrat::mcsat::onecellcad::recursive::DegreeAscending	584
smrat::analyzer::DegreeCollector	584
smrat::mcsat::onecellcad::recursive::DegreeDescending	585
smrat::cadcells::datastructures::DelineatedDerivation< Properties >	585
smrat::cadcells::datastructures::Delineation	588
smrat::cadcells::datastructures::DelineationInterval	589
smrat::mcsat::icp::Dependencies	590
smrat::cadcells::datastructures::DerivationRef< Properties >	591
smrat::mcsat::onecellcad::recursive::DontCoverNullification	594
smrat::cad::debug::DotSubgraph	594
Minisat::DoubleRange	596

smtrat::DynamicPriorityQueue< T, Compare >	598
smtrat::dynarray< T >	600
smtrat::dynarray< implicit_edge_info * >	600
smtrat::dynarray< ineq_edge_info * >	600
smtrat::dynarray_allocator< T >	604
smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >	604
smtrat::cad::ProjectionGlobalInformation::ECData	606
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge	606
smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >	608
smtrat::EQModule< Settings >::graph_info::edge_list_type< edge_info >	608
smtrat::EQModule< Settings >::graph_info::edge_list_type< implicit_edge_info >	608
smtrat::EQModule< Settings >::graph_info::edge_list_type< ineq_edge_info >	608
smtrat::ElementWithOrigins< Element, Origin >	610
smtrat::subtropical::Encoding	639
smtrat::EQGraph< VertexName >	640
smtrat::EQGraph< carl::UVariable >	640
Minisat::Equal< K >	696
Minisat::Equal< CRef >	696
delta::ErrorHandler	696
smtrat::parser::ErrorHandler	696
smtrat::uf_impl::estimator< IteratorType, IsRandomAccess >	724
smtrat::uf_impl::estimator< IteratorType, false >	725
smtrat::execution::ExecutionState	727
smtrat::mcsat::fm::Explanation< Settings >	736
smtrat::mcsat::icp::Explanation	737
smtrat::mcsat::nlsat::Explanation	737
smtrat::mcsat::onecell::Explanation	738
smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 >	738
smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 >	738
smtrat::mcsat::vs::Explanation	739
smtrat::mcsat::nlsat::ExplanationGenerator	739
smtrat::mcsat::vs::ExplanationGenerator< Settings >	740
smtrat::expression::Expression	740
smtrat::expression::ExpressionContent	743
smtrat::parser::ParserState::ExpressionScope	748
std::false_type	
smtrat::is_variant< T >	1057
smtrat::mcsat::FastParallelExplanation< Backends >	750
smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >	751
smtrat::impl::fixedsize_allocator_Impl< T, SizeShift, LargeEnough, SmallEnough >	754
fixedsize_allocator_Impl< T, roundup_log2(Size),(Size >=sizeof(void *)),(Size <=2048)>	
smtrat::impl::fixedsize_allocator_size_helper< T, Size >	756
smtrat::impl::fixedsize_allocator_Impl< T, SizeShift, true, true >	754
impl::fixedsize_allocator_size_helper< T,(sizeof(void *) > sizeof(T) ? sizeof(void *) :sizeof(T))>	
smtrat::fixedsize_allocator< T >	752
smtrat::fixedsize_freelist< ChunkSizeShift >	756
smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, LowEnough >	757
smtrat::impl::fixedsize_freelists< LOWER_SIZE_BOUND, UPPER_SIZE_BOUND >	757
smtrat::impl::fixedsize_allocator_freeLists	753
smtrat::impl::fixedsize_allocator_freeLists< SizeCurrent *2, SizeMax >	757
smtrat::impl::fixedsize_freeLists< SizeCurrent, SizeMax, true >	757
smtrat::parser::FixedWidthConstant< T >	757
formula_visitor	
smtrat::CollectBoolsInUEQs	461
smtrat::ICPModule< Settings >::formulaPtrComp	764
smtrat::FormulaWithOrigins	764
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >	767
smtrat::qe::fm::FourierMotzkinQE	796

smtrat::freelist< T >	823
smtrat::freelist< bucket >	823
smtrat::freelist< edge_info >	823
smtrat::freelist< EdgeType >	823
smtrat::freelist< entry >	823
smtrat::freelist< implicit_edge_info >	823
smtrat::freelist< ineq_edge_info >	823
smtrat::mcsat::FullParallelExplanation< Backends >	824
smtrat::cad::FullSampleComparator< Iterator, Strategy >	825
smtrat::parser::FunctionInstantiator	825
smtrat::parser::UserFunctionInstantiator	1897
smtrat::GBModuleState< Settings >	857
boost::spirit::qi::grammar	
smtrat::parser::ScriptParser< smtrat::parser::SMTLIBParser >	1614
delta::Skipper	1635
delta::SymbolParser	1749
smtrat::parser::AttributeParser	253
smtrat::parser::AttributeValueParser	254
smtrat::parser::BinaryParser	312
smtrat::parser::HexadecimalParser	892
smtrat::parser::IdentifierParser	955
smtrat::parser::KeywordParser	1061
smtrat::parser::QEParser	1546
smtrat::parser::QualifiedIdentifierParser	1547
smtrat::parser::SExpressionParser	1627
smtrat::parser::ScriptParser< Callee >	1614
smtrat::parser::SimpleSymbolParser	1631
smtrat::parser::Skipper	1635
smtrat::parser::SortParser	1652
smtrat::parser::SortedVariableParser	1651
smtrat::parser::SpecConstantParser	1652
smtrat::parser::StringParser	1714
smtrat::parser::SymbolParser	1750
smtrat::parser::TermParser	1802
Minisat::Hash< K >	886
smtrat::EQModule< Settings >::not_asserted_equality::hash	886
std::hash< const smtrat::expression::ExpressionContent * >	886
std::hash< smtrat::expression::BinaryExpression >	887
std::hash< smtrat::expression::Expression >	887
std::hash< smtrat::expression::ITEExpression >	887
std::hash< smtrat::expression::NaryExpression >	888
std::hash< smtrat::expression::QuantifierExpression >	888
std::hash< smtrat::expression::UnaryExpression >	888
std::hash< smtrat::lra::Variable< T1, T2 > >	888
std::hash< smtrat::vs::Substitution >	889
std::hash< std::vector< T > >	889
std::hash_combiner	889
Minisat::Heap< Comp >	890
Minisat::Heap< VarOrderLt >	890
smtrat::icp::HistoryNode	893
smtrat::icp::IcpVariable	950
smtrat::icp::icpVariableComp	954
smtrat::parser::Identifier	954
smtrat::cad::debug::IDSanitizer	956
smtrat::mcsat::fm::IgnoreCoreSettings	956
Implementation	
smtrat::UnionFindInterface< T, Implementation >	1866
smtrat::parser::IndexedFunctionInstantiator	984

smrat::cadcells::datastructures::IndexedRoot	984
smrat::cadcells::datastructures::IndexedRootOrdering	985
smrat::cadcells::datastructures::IndexedRootRelation	987
smrat::InequalitiesTable< Settings >	987
smrat::lra::Bound< T1, T2 >::Info	990
smrat::mcsat::InformationGetter	992
smrat::parser::InstructionHandler	995
smrat::Executor< Strategy >	730
smrat::parseformula::FormulaCollector	758
Minisat::Int64Range	1001
smrat::mcsat::icp::IntervalPropagation	1054
Minisat::IntRange	1056
smrat::is_sample_outside< S >	1056
smrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT >	1057
smrat::expression::ITEExpression	1057
std::iterator	
benchmax::RandomizationAdaptor< T >::iterator	1058
smrat::ModuleInput::IteratorCompare	1059
benchmax::Jobs	1059
smrat::JunctorMerger	1060
Minisat::lbool	1061
smrat::mcsat::onecell::LDBFilteredAllSelectiveSettings	1062
smrat::mcsat::onecell::LDBSettings	1063
smrat::lra::Tableau< Settings, T1, T2 >::LearnedBound	1063
smrat::Module::Lemma	1065
Minisat::LessThan_default< T >	1066
smrat::cad::projection_compare::level	1066
smrat::cad::sample_compare::level	1066
smrat::cad::ProjectionLevelInformation::LevelInfo	1066
smrat::LevelWiseInformation< Settings >	1070
smrat::cad::LiftingTree< Settings >	1072
smrat::ICPModule< Settings >::linearVariable	1076
smrat::mcsat::ClauseChain::Link	1076
std::list< T >	
smrat::ModuleInput	1238
Minisat::Lit	1077
smrat::Manager	1140
Minisat::Map< K, D, H, E >	1148
std::map< K, T >	
smrat::VariantMap< std::string, Value >	1932
smrat::VariantMap< Key, Value >	1932
Minisat::Map< CRef, T, CRefHash >	1148
smrat::mcsat::fm::MaxSizeComparator	1153
smrat::MaxSMT< Solver, Strategy >	1153
smrat::MaxSMT< Strategy, MaxSMTStrategy::LINEAR_SEARCH >	1153
smrat::maxsmt::MaxSMTBackend< Solver, Strategy >	1154
smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL >	1154
smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >	1155
smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >	1155
smrat::mcsat::MCSATBackend< Settings >	1182
smrat::mcsat::MCSATBackend< typename Settings::MCSATSettings >	1182
smrat::mcsat::MCSATMixin< Settings >	1184
smrat::mcsat::MCSATMixin< typename Settings::MCSATSettings >	1184
smrat::mcsat::fm::MinSizeComparator	1196
smrat::mcsat::fm::MinVarCountComparator	1197
smrat::cad::MISGeneration< heuristic >	1197
smrat::Module	1211
smrat::ICPModule< ICPSettings4 >	923

smrat::ICPModule< ICPSettings1 >	923
smrat::LRAModule< LRASettingsICP >	1082
smrat::LRAModule< LRASettings1 >	1082
smrat::BVMModule< Settings >	344
smrat::CSplitModule< Settings >	496
smrat::CoCoAGBModule< Settings >	434
smrat::CubeLIAModule< Settings >	523
smrat::CurryModule< Settings >	550
smrat::EQModule< Settings >	642
smrat::EQPreprocessingModule< Settings >	668
smrat::FouMoModule< Settings >	770
smrat::GBModule< Settings >	826
smrat::ICPModule< Settings >	923
smrat::IncWidthModule< Settings >	956
smrat::IntBlastModule< Settings >	1001
smrat::IntEqModule< Settings >	1027
smrat::LRAModule< Settings >	1082
smrat::NRAILModule< Settings >	1312
smrat::NewCADModule< Settings >	1251
smrat::NewCoveringModule< Settings >	1277
smrat::PBGaussModule< Settings >	1364
smrat::PBPPModule< Settings >	1391
smrat::PModule	1446
smrat::BEModule< Settings >	278
smrat::CNFerModule	405
smrat::EMModule< Settings >	611
smrat::ESModule< Settings >	697
smrat::FPPModule< Settings >	797
smrat::GBPPModule< Settings >	858
smrat::ICEModule< Settings >	897
smrat::LVEModule< Settings >	1114
smrat::MCBModule< Settings >	1155
smrat::PFEModule< Settings >	1420
smrat::SplitSOSModule< Settings >	1653
smrat::SymmetryModule< Settings >	1750
smrat::SATModule< Settings >	1582
smrat::STropModule< Settings >	1715
smrat::UFCegarModule< Settings >	1830
smrat::UnionFindModule< Settings >	1867
smrat::VSMModule< Settings >	1977
smrat::settings::ModuleSettings	1244
smrat::ModuleWrapper< M >	1246
smrat::ModuleWrapper< smrat::EQModule< EQSettingsForPreprocessing > >	1246
smrat::subtropical::Moment	1247
smrat::expression::NaryExpression	1248
smrat::NNFPreparation	1305
smrat::NNFRewriter	1305
benchmax::ssh::Node	1307
delta::Node	1308
delta::NodePrinter< pretty >	1311
smrat::mcsat::onecellcad::recursive::NoHeuristic	1312
smrat::lra::Numeric	1339
smrat::execution::Objective	1346
Minisat::OccLists< Idx, Vec, Deleted >	1346
Minisat::OccLists< Minisat::Lit, Minisat::vec< Minisat::Watcher >, WatcherDeleted >	1346
smrat::mcsat::onecellcad::OneCellCAD	1347
smrat::mcsat::onecellcad::levelwise::LevelwiseCAD	1068
smrat::mcsat::onecellcad::recursive::RecursiveCAD	1554

benchmax::settings::OperationSettings	1349
smtrat::Optimization< Solver >	1350
smtrat::Optimization< Strategy >	1350
Minisat::Option	1351
Minisat::BoolOption	323
Minisat::DoubleOption	595
Minisat::IntOption	1054
Minisat::StringOption	1713
Minisat::Option::OptionLt	1352
smtrat::cad::Origin	1354
smtrat::cad::preprocessor::Origins	1356
smtrat::OrPathShortener	1357
Minisat::OutOfMemoryException	1358
smtrat::parser::OutputWrapper	1358
Minisat::Map< K, D, H, E >::Pair	1358
std::pair	
smtrat::datastructures::OrderPair< T1, T2, reversed >	1353
smtrat::datastructures::OrderPair< T1, T2, true >	1353
smtrat::pairhash< F, S, FirstHasher, SecondHasher >	1359
smtrat::mcsat::ParallelExplanation< Backends >	1359
delta::Parser	1360
smtrat::parser::ParserSettings	1361
smtrat::parser::ParserState	1361
smtrat::PersistentUnionFind	1418
smtrat::cadcells::operators::properties::poly_additional_root_outside	1472
smtrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv	1473
smtrat::cadcells::operators::properties::poly_irreducible_sgn_inv	1474
smtrat::cadcells::operators::properties::poly_ord_inv	1474
smtrat::cadcells::operators::properties::poly_ord_inv_base	1475
smtrat::cadcells::operators::properties::poly_pdel	1476
smtrat::cadcells::operators::properties::poly_semi_sgn_inv	1476
smtrat::cadcells::operators::properties::poly_sgn_inv	1477
smtrat::cadcells::representation::util::PolyDelineation	1478
smtrat::cadcells::representation::util::PolyDelineations	1478
smtrat::cad::ProjectionPolynomialInformation::PolyInfo	1479
smtrat::cad::PolynomialComparator< PolynomialGetter >	1479
smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >	1479
smtrat::cad::PolynomialLiftingQueue< smtrat::cad::BaseProjection >	1479
smtrat::cadcells::datastructures::PolyPool	1481
smtrat::cadcells::datastructures::detail::PolyProperties	1482
smtrat::cadcells::datastructures::PolyRef	1483
smtrat::PolyTree	1483
smtrat::PolyTreeContent	1484
smtrat::cad::Preprocessor	1486
smtrat::cad::PreprocessorSettings	1487
benchmax::settings::PresetSettings	1488
std::priority_queue< T >	
smtrat::PriorityQueue< Iterator, Comparator >	1488
smtrat::PriorityQueue< QueueEntry, smtrat::cad::PolynomialComparator >	1488
smtrat::PriorityQueue< QueueEntry, ProjectionCandidateComparator >	1488
smtrat::PriorityQueue< T, Compare >	1488
delta::Producer	1490
delta::ProgressBar	1491
smtrat::cad::projection_compare::ProjectionComparator< Strategy >	1531
smtrat::cad::projection_compare::ProjectionComparator< Settings::projectionComparator >	1531
smtrat::cad::projection_compare::ProjectionComparator_impl< Args >	1533
smtrat::cad::projection_compare::ProjectionComparator_impl< degree, It >	1533
smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D >	1531

smrat::cad::projection_compare::ProjectionComparator_impl< level, gt, degree, lt >	1533
smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD >	1532
smrat::cad::projection_compare::ProjectionComparator_impl< level, lt, degree, lt >	1533
smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID >	1532
smrat::cad::projection_compare::ProjectionComparator_impl< type, gt, degree, lt >	1533
smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD >	1532
smrat::cad::projection_compare::ProjectionComparator_impl< type, lt, degree, lt >	1533
smrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD >	1533
smrat::cad::ProjectionGlobalInformation	1533
smrat::cad::ProjectionInformation	1535
smrat::cad::ProjectionLevelInformation	1537
smrat::cad::ProjectionOperator	1537
smrat::cad::ProjectionPolynomialInformation	1538
smrat::cadcells::datastructures::Projections	1539
smrat::cadcells::operators::mccallum_filtered_Impl::PropertiesSet	1542
smrat::cadcells::operators::PropertiesSet< Op >	1542
smrat::cadcells::operators::PropertiesSet< op::mccallum >	1542
smrat::cadcells::datastructures::PropertiesT< Ts >	1542
smrat::cadcells::datastructures::PropertiesT< Ts... >	1542
smrat::cadcells::datastructures::PropertiesT< T, Ts... >	1543
smrat::cadcells::datastructures::PropertiesTContent< T, is_flag >	1543
smrat::cadcells::datastructures::PropertiesTContent< T, false >	1543
smrat::cadcells::datastructures::PropertiesTContent< T, T::is_flag >	1543
smrat::cadcells::datastructures::PropertiesTContent< T, true >	1543
smrat::cadcells::datastructures::property_hash< T >	1544
smrat::PseudoBoolEncoder	1544
smrat::CardinalityEncoder	387
smrat::ExactlyOneCommanderEncoder	725
smrat::LongFormulaEncoder	1081
smrat::MixedSignEncoder	1198
smrat::RNSEncoder	1565
smrat::ShortFormulaEncoder	1628
smrat::TotalizerEncoder	1823
smrat::PseudoBoolNormalizer	1545
smrat::expression::QuantifierExpression	1547
Minisat::Queue< T >	1548
smrat::mcsat::icp::QueueEntry	1549
benchmax::RandomizationAdaptor< T >	1550
smrat::RationalCapsule	1551
boost::spirit::qi::real_parser	
smrat::parser::DecimalParser	582
boost::spirit::qi::real_policies	
smrat::parser::RationalPolicies	1552
smrat::mcsat::onecellcad::RealAlgebraicPoint< Number >	1552
smrat::mcsat::onecellcad::RealAlgebraicPoint< Rational >	1552
smrat::fixedsize_allocator< T >::rebind< U >	1554
Minisat::RegionAllocator< T >	1559
Minisat::RegionAllocator< uint32_t >	1559
Minisat::ClauseAllocator	398
smrat::NNFRewriter::remove_xor_first_arg	1561
smrat::ReplaceVariablesRewriter	1561
delta::ErrorHandler::result< typename >	1563
smrat::parser::ErrorHandler::result< typename >	1563
smrat::cad::preprocessor::ResultantRule	1563
benchmax::Results	1564
smrat::cadcells::operators::properties::root_ordering_holds	1566

smrat::cadcells::operators::properties::root_well_def	1567
smrat::cadcells::datastructures::RootFunction	1568
smrat::mcsat::arithmetic::RootIndexer< RANT >	1570
smrat::mcsat::arithmetic::RootIndexer< typename Poly::RootType >	1570
smrat::mcsat::arithmetic::RootIndexer< typename Polynomial::RootType >	1570
smrat::cad::Sample	1570
smrat::cad::sample_compare::SampleComparator< Iterator, Strategy >	1573
smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T >	825
smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type >	825
smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value >	825
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T >	1573
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >	1573
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >	1576
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value >	1573
smrat::cad::sample_compare::SampleComparator_impl< It, Args >	1577
smrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, size, It >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LS >	1573
smrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LT >	1573
smrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, absvalue, It >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA >	1573
smrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, size, It >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS >	1574
smrat::cad::sample_compare::SampleComparator_impl< Iterator, level, gt, type, gt, size, It, absvalue, It >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA >	1574
smrat::cad::sample_compare::SampleComparator_impl< Iterator, size, It >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S >	1574
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value >	1576
smrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T >	1575
smrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, level, gt, size, It, absvalue, It >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA >	1575
smrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, size, It >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TS >	1575
smrat::cad::sample_compare::SampleComparator_impl< Iterator, type, gt, size, It, absvalue, It >	1577
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA >	1576
smrat::cadcells::datastructures::SampledDerivation< Properties >	1577
smrat::SampledDerivationRefCompare	1580
smrat::cad::SampleIteratorQueue< Iterator, Comparator >	1580
smrat::cad::SampleIteratorQueue< Iterator, smrat::cad::FullSampleComparator >	1580
smrat::cad::SampleIteratorQueue< Iterator, smrat::cad::sample_compare::SampleComparator >	1580
smrat::sampling< S >	1582
smrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING >	1582
smrat::parser::ParserState::ScriptScope	1616
smrat::mcsat::onecellcad::Section	1617
smrat::onecellcad::recursive::Section	1617
smrat::mcsat::onecellcad::levelwise::SectionHeuristic1	1618
smrat::mcsat::onecellcad::levelwise::SectionHeuristic2	1618
smrat::mcsat::onecellcad::levelwise::SectionHeuristic3	1618
smrat::mcsat::onecellcad::Sector	1619
smrat::onecellcad::recursive::Sector	1619
smrat::mcsat::onecellcad::levelwise::SectorHeuristic1	1620
smrat::mcsat::onecellcad::levelwise::SectorHeuristic2	1620
smrat::mcsat::onecellcad::levelwise::SectorHeuristic3	1621
smrat::subtropical::Separator	1621

smrat::mcsat::SequentialAssignment< Backends >	1622
smrat::mcsat::SequentialExplanation< Backends >	1622
carl::settings::Settings	
benchmax::settings::Settings	1622
smrat::settings::Settings	1626
delta::Settings	1622
carl::settings::SettingsParser	
benchmax::SettingsParser	1627
smrat::SettingsParser	1627
smrat::CNFerModule::SettingsType	1627
benchmax::simple_parser	1630
smrat::parser::Theories::SimpleSortAdder	1631
smrat::expression::simplifier::Simplifier	1632
smrat::expression::simplifier::SimplifierChainCaller< chainID >	1632
smrat::expression::simplifier::SimplifierChainCaller< 0 >	1633
carl::Singleton	
benchmax::SettingsParser	1627
benchmax::settings::Settings	1622
smrat::LOG	1080
smrat::MonomialMappingByVariablePool	1247
smrat::PolyTreePool	1485
smrat::SettingsComponents	1626
smrat::SettingsParser	1627
smrat::cadcells::representation::approximation::ApxCriteria	240
smrat::expression::ExpressionPool	747
smrat::resource::Limiter	1075
smrat::settings::Settings	1626
smrat::validation::ValidationCollector	1898
smrat::cad::sample_compare::size	1635
benchmax::settings::SlurmBackendSettings	1638
smrat::parser::SMTLIBParser	1639
smrat::execution::SoftAssertion	1650
smrat::settings::SolverSettings	1650
benchmax::settings::SSHBackendSettings	1682
benchmax::ssh::SSHConnection	1683
smrat::vs::State	1685
boost::static_visitor	
smrat::parser::conversion::VariantConverter< types::BVTerm >	1931
smrat::parser::conversion::VariantVariantConverter< types::AttributeValue >	1934
smrat::parser::conversion::VariantVariantConverter< types::TermType >	1934
smrat::expression::ExpressionConverter	744
smrat::expression::ExpressionModifier	745
smrat::expression::ExpressionTypeChecker< T >	749
smrat::expression::ExpressionVisitor	749
smrat::expression::simplifier::BaseSimplifier	271
smrat::expression::simplifier::DuplicateSimplifier	597
smrat::expression::simplifier::MergeSimplifier	1190
smrat::expression::simplifier::NegationSimplifier	1249
smrat::expression::simplifier::SingletonSimplifier	1633
smrat::parser::Instantiator< V, T >	993
smrat::parser::conversion::VariantConverter< Res >	1931
smrat::parser::conversion::VariantVariantConverter< Res >	1934
smrat::variant_hash_visitor	1930
smrat::StaticUnionFind	1710
Statistics	
smrat::Module::ModuleStatistics	1245
smrat::analyzer::AnalyzerStatistics	239
smrat::statistics::StatisticsSettings	1711

smrat::StrategyGraph	1711
delta::String	1712
benchmax::slurm::SubmitfileProperties	1742
smrat::vs::Substitution	1743
smrat::vs::substitutionPointerEqual	1747
smrat::vs::substitutionPointerHash	1747
smrat::cadcells::datastructures::SymbolicInterval	1748
boost::spirit::qi::symbols	
smrat::parser::LogicParser	1081
smrat::parser::QuantifierParser	1548
smrat::lra::Tableau< Settings, T1, T2 >	1778
smrat::lra::Tableau< typename Settings::Tableau_settings, LRABoundType, LRAEntryType >	1778
smrat::lra::TableauEntry< T1, T2 >	1796
smrat::lra::TableauEntry< LRABoundType, LRAEntryType >	1796
smrat::cadcells::datastructures::TaggedIndexedRoot	1799
smrat::mcsat::onecellcad::TagPoly	1799
smrat::Task	1800
delta::TempFilenameGenerator	1801
smrat::mcsat::vs::helper::TestCandidate	1803
smrat::parser::Theories	1804
smrat::parser::TheoryError	1807
smrat::mcsat::TheoryLevel	1807
smrat::Module::TheoryPropagation	1808
smrat::ThreadPool	1813
smrat::cad::debug::TikzBasePrinter	1813
smrat::cad::debug::TikzDAGPrinter	1815
smrat::cad::debug::TikzTreePrinter	1817
smrat::cad::debug::TikzHistoryPrinter	1816
benchmax::Tool	1819
benchmax::MathSAT	1150
benchmax::Minisat	1191
benchmax::Minisatp	1194
benchmax::SMTRAT	1642
benchmax::SMTRAT_OPB	1647
benchmax::SMTRAT_Analyzer	1645
benchmax::Z3	2007
benchmax::settings::ToolSettings	1822
smrat::TotalizerTree	1825
smrat::cad::projection_compare::type	1826
smrat::cad::sample_compare::type	1826
smrat::uf_impl::uf_data_wrapper< T, compression >	1826
smrat::uf_impl::uf_data_wrapper< void, compression >	1827
smrat::uf_impl::uf_rank_and_class< compression >	1828
smrat::uf_impl::uf_rank_and_class< true >	1829
smrat::UFRewriter	1856
boost::spirit::qi::uint_parser	
smrat::parser::NumeralParser	1339
smrat::expression::UnaryExpression	1857
smrat::cad::debug::UnifiedData	1857
smrat::parser::types::UninterpretedTheory	1858
smrat::union_find< T, no_compression, MaxTracebackLength >	1862
smrat::union_find< g_iterator, !Settings::uf_use_compression, Settings::uf_max_traceback_length >	1862
smrat::union_find< uf_component_entry, !Settings::uf_use_compression, Settings::uf_max_traceback_length >	1862
smrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher	1894
smrat::UnsatCore< Solver, Strategy >	1894
smrat::UnsatCore< Strategy, UnsatCoreStrategy::ModelExclusion >	1894
smrat::unsatcore::UnsatCoreBackend< Solver, Strategy >	1895

smrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >	1895
smrat::vs::unsignedTripleCmp	1896
smrat::validation::ValidationPoint	1898
smrat::validation::ValidationPrinter< SOF >	1900
smrat::validation::ValidationSettings	1900
smrat::lra::Value< T >	1900
smrat::lra::Value< LRABoundType >	1900
smrat::lra::Value< T1 >	1900
smrat::lra::Variable< T1, T2 >	1908
smrat::vb::Variable< T >	1918
smrat::lra::Variable< LRABoundType, LRAEntryType >	1908
smrat::vb::VariableBounds< T >	1921
smrat::vb::VariableBounds< ConstraintT >	1921
smrat::vb::VariableBounds< FormulaT >	1921
smrat::VariableCapsule	1928
smrat::mcsat::variableordering::VariableIDs	1929
smrat::VariableRewriteRule	1929
smrat::VarSchedulerBase	1935
smrat::VarSchedulerMcsatBase	1945
smrat::TheoryVarSchedulerStatic< vot >	1809
smrat::VarSchedulerMcsatActivityPreferTheory< vot >	1941
smrat::VarSchedulerMcsatBooleanFirst< vot >	1948
smrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >	1952
smrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >	1955
smrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >	1959
smrat::VarSchedulerMinisat	1963
smrat::VarSchedulerFixedRandom	1938
smrat::VarSchedulerRandom	1966
smrat::VarSchedulerSMTtheoryGuided< theory_conflict_guided_decision_heuristic >	1969
Minisat::vec< T >	1972
Minisat::vec< bool >	1972
Minisat::vec< char >	1972
Minisat::vec< double >	1972
Minisat::vec< Idx >	1972
Minisat::vec< int >	1972
Minisat::vec< Minisat::CRef >	1972
Minisat::vec< Minisat::lbool >	1972
Minisat::vec< Minisat::Lit >	1972
Minisat::vec< Minisat::Map::Pair >	1972
Minisat::vec< Minisat::vec< Minisat::Lit > >	1972
Minisat::vec< std::pair< Abstraction *, Abstraction * > >	1972
Minisat::vec< unsigned >	1972
Minisat::vec< VarData >	1972
Minisat::vec< Vec >	1972
std::vector< T >	
smrat::cad::projection::Reducta< Poly >	1558
smrat::parser::SExpressionSequence< T >	1628
smrat::parser::conversion::VectorVariantConverter< Res >	1975
smrat::parser::conversion::VectorVariantConverter< types::BVTerm >	1975
smrat::subtropical::Vertex	1976
Minisat::Watcher	2005
smrat::ICPModule< Settings >::weights	2006
benchmax::XMLWriter	2006

0.13 Data Structure Index

0.13.1 Data Structures

Here are the data structures with brief descriptions:

<code>smrat::AbstractModuleFactory</code>	235
<code>smrat::parser::AbstractTheory</code>	
Base class for all theories	235
<code>smrat::cad::sample_compare::absvalue</code>	238
<code>smrat::analyzer::AnalysisSettings</code>	238
<code>smrat::analyzer::AnalyzerStatistics</code>	239
<code>smrat::AnnotatedBVTerm</code>	239
<code>smrat::cadcells::representation::approximation::ApxCriteria</code>	240
<code>smrat::cadcells::representation::approximation::ApxSettings</code>	241
<code>benchmax::slurm::ArchiveProperties</code>	
All properties needed to archive log files	243
<code>smrat::parser::ArithmeticTheory</code>	
Implements the theory of arithmetic, including LRA, LIA, NRA and NIA	244
<code>smrat::parser::types::ArithmeticTheory</code>	
Types of the arithmetic theory	246
<code>smrat::execution::Assertion</code>	247
<code>smrat::cad::preprocessor::AssignmentCollector</code>	247
<code>smrat::mcsat::arithmetic::AssignmentFinder</code>	248
<code>smrat::mcsat::smtaf::AssignmentFinder< Settings ></code>	249
<code>smrat::mcsat::arithmetic::AssignmentFinder_ctx</code>	249
<code>smrat::mcsat::arithmetic::AssignmentFinder_detail</code>	250
<code>smrat::mcsat::smtaf::AssignmentFinder_SMT</code>	251
<code>smrat::cadcells::datastructures::detail::AssignmentProperties</code>	251
<code>smrat::parser::Attribute</code>	
Represents an <code>Attribute</code>	252
<code>smrat::parser::AttributeParser</code>	253
<code>smrat::parser::AttributeValueParser</code>	254
<code>smrat::AxiomFactory</code>	254
<code>benchmax::Backend</code>	
Base class for all backends	255
<code>smrat::Backend< Settings ></code>	257
<code>smrat::BackendLink</code>	260
<code>smrat::BackendSynchronisation</code>	261
<code>smrat::Backtrackable< UnionFind ></code>	261
<code>smrat::cadcells::datastructures::BaseDerivation< Properties ></code>	
A <code>BaseDerivation</code> has a level and a set of properties of this level, and an underlying derivation representing the lower levels	263
<code>smrat::cad::BaseProjection< Settings ></code>	265
<code>smrat::cad::BaseSettings</code>	270
<code>smrat::expression::simplifier::BaseSimplifier</code>	271
<code>smrat::cad::Origin::BaseType</code>	273
<code>smrat::mcsat::onecell::BCApproximationSettings</code>	275
<code>smrat::mcsat::onecell::BCFilteredAllSelectiveSettings</code>	275
<code>smrat::mcsat::onecell::BCFilteredAllSettings</code>	276
<code>smrat::mcsat::onecell::BCFilteredBoundsSettings</code>	276
<code>smrat::mcsat::onecell::BCFilteredSamplesSettings</code>	276
<code>smrat::mcsat::onecell::BCFilteredSettings</code>	277
<code>smrat::mcsat::onecell::BCSettings</code>	277
<code>smrat::BEModule< Settings ></code>	278
<code>benchmax::BenchmarkResult</code>	
Results for a single benchmark run	305
<code>benchmax::BenchmarkSet</code>	
A set of benchmarks from some common base directory	307

benchmax::settings::BenchmarkSettings	308
Settings for benchmarks	308
smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >	
Container that stores expensive to construct objects and allows the fast lookup with respect to two independent keys within the objects	309
smtrat::expression::BinaryExpression	311
smtrat::datastructures::BinaryHeap< T >	312
smtrat::parser::BinaryParser	
Parses binaries: #b [01] +	312
smtrat::parser::BitvectorTheory	
Implements the theory of bitvectors	313
smtrat::parser::types::BitvectorTheory	
Types of the theory of bitvectors	316
smtrat::BlastedConstr	317
smtrat::BlastedPoly	318
smtrat::mcsat::Bookkeeping	
Represent the trail, i.e	319
smtrat::parser::BooleanEncodingTheory	
Implements the theory of bitvectors	320
Minisat::BoolOption	323
smtrat::BoolUEQRewriter	324
smtrat::cadcells::datastructures::Bound	
Bound type of a SymbolicInterval	325
smtrat::lra::Bound< T1, T2 >	
Stores a bound, which could be an upper " $\leq b$ " or a lower bound " $\geq b$ " for a bound value b	326
smtrat::mcsat::fm::Bound	335
smtrat::vb::Bound< T >	
Class for the bound of a variable	336
smtrat::Branching	
Stores all necessary information of an branch, which can be used to detect probable infinite loop of branchings	341
smtrat::BVAnnotation	342
smtrat::BVDirectEncoder	344
smtrat::BVModule< Settings >	344
smtrat::by_address_hasher< IterType >	
Hash an iterator by the pointed-to address	372
smtrat::cad::CAD< Settings >	372
smtrat::qe::cad::CAD< Settings >	375
smtrat::cad::CADConstraints< BT >	377
smtrat::cad::CADCore< CH >	381
smtrat::cad::CADCore< CoreHeuristic::BySample >	381
smtrat::cad::CADCore< CoreHeuristic::EnumerateAll >	381
smtrat::cad::CADCore< CoreHeuristic::Interleave >	382
smtrat::cad::CADCore< CoreHeuristic::PreferProjection >	382
smtrat::cad::CADCore< CoreHeuristic::PreferSampling >	383
smtrat::qe::cad::CADElimination	383
smtrat::cad::CADPreprocessor	383
smtrat::cad::CADPreprocessorSettings	385
smtrat::qe::cad::CADSettings	386
smtrat::CardinalityEncoder	387
smtrat::cadcells::representation::cell< H >	
Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds	388
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL >	388
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION >	389
smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST_CELL_EW >	389
smtrat::cadcells::representation::cell< CellHeuristic::CHAIN_EQ >	389
smtrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS >	390

smrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ >	390
smrat::cadcells::representation::cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EW >	390
smrat::cadcells::operators::properties::cell_connected	391
smrat::cadcells::representation::approximation::CellApproximator	391
smrat::cadcells::datastructures::CellRepresentation< P >	
Represents a cell	392
delta::Checker	
This class takes care of calling the actual solver	393
benchmax::slurm::ChunkedSubmitfileProperties	
All properties needed to create a submit file	394
Minisat::Clause	396
Minisat::ClauseAllocator	398
smrat::mcsat::ClauseChain	
An explanation is either a single clause or a chain of clauses, satisfying the following properties:	401
smrat::sat::detail::ClauseChecker	403
smrat::CMakeOptionPrinter	404
Minisat::CMap< T >	404
smrat::CNFerModule	405
smrat::CoCoAGBModule< Settings >	434
smrat::CollectBoolsInUEQs	461
smrat::CollectionWithOrigins< Element, Origin >	462
smrat::ICPModule< Settings >::comp	
Typedefs:	464
smrat::cadcells::datastructures::CompoundMax	
Represents the maximum function of the contained indexed root functions	464
smrat::cadcells::datastructures::CompoundMin	
Represents the minimum function of the contained indexed root functions	464
smrat::vs::Condition	465
benchmax::CondorBackend	
Backend for the HTCondor batch system	468
smrat::mcsat::fm::ConflictGenerator< Comparator >	470
smrat::cad::ConflictGraph	
Representation of a bipartite graph (C+S, E) of vertices C and S, representing the constraints and samples, respectively	470
smrat::parser::Theories::ConstantAdder	
Helper functor for <code>addConstants()</code> method	472
smrat::cad::CADConstraints< BT >::ConstraintComparator	473
smrat::cad::preprocessor::ConstraintUpdate	473
smrat::ConstrTree	474
delta::Consumer	
This class takes care of asynchronous execution of calls to the solver	475
smrat::LRAModule< Settings >::Context	
Stores a formula, being part of the received formula of this module, and the position of this formula in the passed formula	476
smrat::icp::ContractionCandidate	477
smrat::icp::contractionCandidateComp	482
smrat::icp::ContractionCandidateManager	482
smrat::parser::conversion::Converter< To >	484
smrat::parser::conversion::Converter< FormulaT >	484
smrat::parser::conversion::Converter< Poly >	485
smrat::parser::conversion::Converter< types::BVTerm >	485
benchmax::settings::CoreSettings	
Core settings	486
smrat::settings::CoreSettings	487
smrat::parser::CoreTheory	
Implements the core theory of the booleans	488
smrat::parser::types::CoreTheory	
Types of the core theory	490

<code>smrat::cadcells::representation::covering< H ></code>	491
<code>smrat::mcsat::arithmetic::Covering</code>	
Semantics: The space is divided into a number of intervals: (-oo,a][a,a](a,b][b,oo) A bit is set if the constraints refutes the corresponding interval	491
<code>smrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING ></code>	493
<code>smrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST_CELL_COVERING_EW ></code>	493
<code>smrat::cadcells::representation::covering< CoveringHeuristic::CHAIN_COVERING ></code>	493
<code>smrat::cadcells::datastructures::CoveringDescription</code>	
Describes a covering of the real line by SymbolicIntervals (given an appropriate sample)	494
<code>smrat::cadcells::datastructures::CoveringRepresentation< P ></code>	
Represents a covering over a cell	494
<code>smrat::mcsat::onecellcad::recursive::CoverNullification</code>	496
<code>smrat::CSplitModule< Settings ></code>	496
<code>smrat::CubeLIAModule< Settings ></code>	523
<code>smrat::CurryModule< Settings ></code>	550
<code>smrat::CycleEnumerator< FHG, Collector ></code>	
This class encapsulates an algorithm for enumerating all cycles	576
<code>benchmax::Database</code>	
Dummy database that effectively disables storing to database. Set BENCHMAX_DATABASE to actually use a database	576
<code>benchmax::DBAL</code>	578
<code>smrat::parser::DecimalParser</code>	
Parses decimals: numeral.0*numeral	582
<code>Minisat::DeepEqual< K ></code>	582
<code>Minisat::DeepHash< K ></code>	582
<code>smrat::mcsat::fm::DefaultComparator</code>	
Does not order anything	582
<code>smrat::mcsat::fm::DefaultSettings</code>	583
<code>smrat::mcsat::smtaf::DefaultSettings</code>	583
<code>smrat::mcsat::vs::DefaultSettings</code>	583
<code>smrat::cad::projection_compare::degree</code>	584
<code>smrat::mcsat::onecellcad::recursive::DegreeAscending</code>	584
<code>smrat::analyzer::DegreeCollector</code>	584
<code>smrat::mcsat::onecellcad::recursive::DegreeDescending</code>	585
<code>smrat::cadcells::datastructures::DelineatedDerivation< Properties ></code>	
A DelineatedDerivation is a BaseDerivation with a <code>Delineation</code> and an underlying <code>SampledDerivation</code>	
585	
<code>smrat::cadcells::datastructures::Delineation</code>	
Represents the delineation of a set of polynomials (at a sample), that is	588
<code>smrat::cadcells::datastructures::DelineationInterval</code>	
An interval of a delineation	589
<code>smrat::mcsat::icp::Dependencies</code>	590
<code>smrat::cadcells::datastructures::DerivationRef< Properties ></code>	
A reference to a derivation, which is either	591
<code>smrat::mcsat::onecellcad::recursive::DontCoverNullification</code>	594
<code>smrat::cad::debug::DotSubgraph</code>	594
<code>Minisat::DoubleOption</code>	595
<code>Minisat::DoubleRange</code>	596
<code>smrat::expression::simplifier::DuplicateSimplifier</code>	597
<code>smrat::DynamicPriorityQueue< T, Compare ></code>	598
<code>smrat::dynarray< T ></code>	
Dynarray is similar to std::vector, but has only a reduced interface	600
<code>smrat::dynarray_allocator< T ></code>	604
<code>smrat::datastructures::DynHeap< KeyType, Compare, WeightType ></code>	604
<code>smrat::cad::ProjectionGlobalInformation::ECData</code>	606
<code>smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge</code>	
Internal type of an edge	606

smrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >	608
Implements a list of edges	
smrat::ElementWithOrigins< Element, Origin >	610
smrat::EMModule< Settings >	611
smrat::subtropical::Encoding	639
smrat::EQGraph< VertexName >	640
smrat::EQModule< Settings >	642
smrat::EQPreprocessingModule< Settings >	668
Minisat::Equal< K >	696
delta::ErrorHandler	696
smrat::parser::ErrorHandler	696
smrat::ESModule< Settings >	697
smrat::uf_impl::estimator< IteratorType, IsRandomAccess >	
Distance estimation for a range delimited by iterators; if the given iterators are random access, returns the result of std::distance in the estimate_distance function	724
smrat::uf_impl::estimator< IteratorType, false >	725
smrat::ExactlyOneCommanderEncoder	725
smrat::execution::ExecutionState	727
smrat::Executor< Strategy >	730
smrat::mcsat::fm::Explanation< Settings >	736
smrat::mcsat::icp::Explanation	737
smrat::mcsat::nlsat::Explanation	737
smrat::mcsat::onecell::Explanation	738
smrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 >	738
smrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 >	738
smrat::mcsat::vs::Explanation	739
smrat::mcsat::nlsat::ExplanationGenerator	739
smrat::mcsat::vs::ExplanationGenerator< Settings >	740
smrat::expression::Expression	740
smrat::expression::ExpressionContent	743
smrat::expression::ExpressionConverter	744
smrat::expression::ExpressionModifier	745
smrat::expression::ExpressionPool	747
smrat::parser::ParserState::ExpressionScope	748
smrat::expression::ExpressionTypeChecker< T >	749
smrat::expression::ExpressionVisitor	749
smrat::mcsat::FastParallelExplanation< Backends >	
This explanation executes all given explanation in parallel processes and waits for the fastest explanation, returning the fastest delivered explanation, terminating all other parallel processes	750
smrat::mcsat::variableordering::detail::FeatureCollector< Objects >	
This class manages features that are used to valuate variables on objects	751
smrat::fixedsize_allocator< T >	
An allocator meant to be used for containers that typically allocate single objects or nodes, such as sets, maps and linked lists	752
smrat::fixedsize_allocator< void >	
I do not know where an allocator for void would be of any use; but in order to mimic std::allocator (where this is explicitly mentioned) as good as possible, we do this for the void case	753
smrat::impl::fixedsize_allocator_freeLists	753
smrat::impl::fixedsize_allocator_Impl< T, SizeShift, LargeEnough, SmallEnough >	754
smrat::impl::fixedsize_allocator_Impl< T, SizeShift, true, true >	754
smrat::impl::fixedsize_allocator_size_helper< T, Size >	756
smrat::fixedsize_freelist< ChunkSizeShift >	
Implements a fixed-(at-compile-time)-size allocator based on a free list embedded in the space of currently unused objects	756
smrat::impl::fixedsize_freeLists< SizeCurrent, SizeMax, LowEnough >	
Publicly derive from all <code>fixedsize_freeList</code> types with chunk sizes between <code>SizeCurrent</code> and <code>SizeMax</code>	757
smrat::impl::fixedsize_freeLists< SizeCurrent, SizeMax, true >	757

smrat::parser::FixedWidthConstant< T >	757
Represents a constant of a fixed width	
smrat::parseformula::FormulaCollector	758
smrat::ICPModule< Settings >::formulaPtrComp	764
smrat::FormulaWithOrigins	
Stores a formula along with its origins	764
smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >	
This class implements a forward hypergraph	767
smrat::FouMoModule< Settings >	
A module which applies the Fourier-Motzkin algorithm	770
smrat::qe::fm::FourierMotzkinQE	
Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints	796
smrat::FPPModule< Settings >	797
smrat::freelist< T >	
Implements a fixed-(at-runtime)-size allocator based on a free list embedded in the space of currently unused objects	823
smrat::mcsat::FullParallelExplanation< Backends >	
This explanation executes all given explanation parallel in multiple threads and waits until every single one has finished, returning the result of the first listed explanation	824
smrat::cad::FullSampleComparator< Iterator, Strategy >	825
smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T >	825
smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type >	825
smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value >	825
smrat::parser::FunctionInstantiator	825
smrat::GBModule< Settings >	
A solver module based on Groebner basis	826
smrat::GBModuleState< Settings >	
A class to save the current state of a GBModule	857
smrat::GBPPModule< Settings >	858
Minisat::Hash< K >	886
smrat::EQModule< Settings >::not_asserted_equality::hash	886
std::hash< const smrat::expression::ExpressionContent * >	886
std::hash< smrat::expression::BinaryExpression >	887
std::hash< smrat::expression::Expression >	887
std::hash< smrat::expression::ITEExpression >	887
std::hash< smrat::expression::NaryExpression >	888
std::hash< smrat::expression::QuantifierExpression >	888
std::hash< smrat::expression::UnaryExpression >	888
std::hash< smrat::lra::Variable< T1, T2 > >	
Implements std::hash for sort value	888
std::hash< smrat::vs::Substitution >	889
std::hash< std::vector< T > >	889
std::hash_combiner	889
Minisat::Heap< Comp >	890
smrat::parser::HexadecimalParser	
Parses hexadecimals: #x [0-9a-fA-F] +	892
smrat::icp::HistoryNode	893
smrat::ICEModule< Settings >	897
smrat::ICPModule< Settings >	923
smrat::icp::IcpVariable	950
smrat::icp::icpVariableComp	954
smrat::parser::Identifier	954
smrat::parser::IdentifierParser	955
smrat::cad::debug::IDSanitizer	956
smrat::mcsat::fm::IgnoreCoreSettings	956
smrat::IncWidthModule< Settings >	956
smrat::parser::IndexedFunctionInstantiator	984

smrat::cadcells::datastructures::IndexedRoot	Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample)	984
smrat::cadcells::datastructures::IndexedRootOrdering	Describes an ordering of IndexedRoots	985
smrat::cadcells::datastructures::IndexedRootRelation	A relation between two roots	987
smrat::InequalitiesTable< Settings >	Datastructure for the GBModule	987
smrat::lra::Bound< T1, T2 >::Info	Stores some additional information for a bound	990
smrat::mcsat::InformationGetter	992
smrat::parser::Instantiator< V, T >	993
smrat::parser::InstructionHandler	995
Minisat::Int64Range	1001
smrat::IntBlastModule< Settings >	1001
smrat::IntEqModule< Settings >	A module which checks whether the equations contained in the received (linear integer) formula have a solution	1027
smrat::mcsat::icp::IntervalPropagation	1054
Minisat::IntOption	1054
Minisat::IntRange	1056
smrat::is_sample_outside< S >	1056
smrat::is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT >	1057
smrat::is_variant< T >	States whether a given type is a boost::variant	1057
smrat::expression::ITEExpression	1057
benchmax::RandomizationAdaptor< T >::iterator	1058
smrat::ModuleInput::IteratorCompare	1059
benchmax::Jobs	Represents a set of jobs, constructed as the cartesian product of a set of tools and a set of benchmarks	1059
smrat::JunctorMerger	1060
smrat::parser::KeywordParser	Parses keywords::simple_symbol	1061
Minisat::Ibool	1061
smrat::mcsat::onecell::LDBFilteredAllSelectiveSettings	1062
smrat::mcsat::onecell::LDBSettings	1063
smrat::lra::Tableau< Settings, T1, T2 >::LearnedBound	1063
smrat::Module::Lemma	1065
Minisat::LessThan_default< T >	1066
smrat::cad::projection_compare::level	1066
smrat::cad::sample_compare::level	1066
smrat::cad::ProjectionLevelInformation::LevelInfo	1066
smrat::mcsat::onecellcad::levelwise::LevelwiseCAD	1068
smrat::LevelWiseInformation< Settings >	1070
smrat::cad::LiftingTree< Settings >	1072
smrat::resource::Limiter	1075
smrat::ICPModule< Settings >::linearVariable	1076
smrat::mcsat::ClauseChain::Link	1076
Minisat::Lit	1077
benchmax::LocalBackend	This backend simply runs files sequentially on the local machine	1078
smrat::LOG	1080
smrat::parser::LogicParser	1081
smrat::LongFormulaEncoder	1081
smrat::LRAModule< Settings >	A module which performs the Simplex method on the linear part of it's received formula	1082

smrat::LVEModule< Settings >	1114
smrat::Manager	
Base class for solvers	1140
Minisat::Map< K, D, H, E >	1148
benchmax::MathSAT	
Tool wrapper for MathSAT for SMT-LIB	1150
smrat::mcsat::fm::MaxSizeComparator	
This heuristic chooses the explanation excluding the largest interval	1153
smrat::MaxSMT< Solver, Strategy >	1153
smrat::maxsmt::MaxSMTBackend< Solver, Strategy >	1154
smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU_MALIK_INCREMENTAL >	1154
smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >	1155
smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >	1155
smrat::MCBModule< Settings >	1155
smrat::mcsat::MCSATBackend< Settings >	1182
smrat::mcsat::MCSATMixin< Settings >	1184
smrat::expression::simplifier::MergeSimplifier	1190
benchmax::Minisat	
Tool wrapper for a Minisat solver	1191
benchmax::Minisatp	
Tool wrapper for the Minisatp solver for pseudo-Boolean problems	1194
smrat::mcsat::fm::MinSizeComparator	
This heuristic chooses the explanation excluding the smallest interval	1196
smrat::mcsat::fm::MinVarCountComparator	
This heuristic tries to minimize the number of variables occuring in the explanation	1197
smrat::cad::MISGeneration< heuristic >	1197
smrat::MixedSignEncoder	1198
smrat::cad::ModelBasedProjection< incrementality, backtracking, Settings >	1199
smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >	
This class implements a projection that supports no incrementality and expects backtracking to be in order	1205
smrat::Module	
A base class for all kind of theory solving methods	1211
smrat::ModuleFactory< Module >	1238
smrat::ModuleInput	
The input formula a module has to consider for it's satisfiability check	1238
smrat::settings::ModuleSettings	1244
smrat::Module::ModuleStatistics	1245
smrat::ModuleWrapper< M >	1246
smrat::subtropical::Moment	
Represents the normal vector component and the sign variable assigned to a variable in an original constraint	1247
smrat::MonomialMappingByVariablePool	1247
smrat::expression::NaryExpression	1248
smrat::expression::simplifier::NegationSimplifier	1249
smrat::NewCADModule< Settings >	1251
smrat::NewCoveringModule< Settings >	1277
smrat::NNFPreparation	1305
smrat::NNFRewriter	1305
benchmax::ssh::Node	
Specification of a computation node for the SSH backend	1307
delta::Node	
This class represents a node in a SMTLIB file	1308
delta::NodePrinter< pretty >	1311
smrat::mcsat::onecellcad::recursive::NoHeuristic	1312
smrat::NRAILModule< Settings >	1312
smrat::parser::NumeralParser	
Parses numerals: (0 [1-9] [0-9]*)	1339

smrat::ira::Numeric	1339
smrat::execution::Objective	1346
Minisat::OccLists< Idx, Vec, Deleted >	1346
smrat::mcsat::onecellcad::OneCellCAD	1347
benchmax::settings::OperationSettings Operation settings	1349
smrat::Optimization< Solver >	1350
Minisat::Option	1351
Minisat::Option::OptionLt	1352
smrat::datastructures::OrderPair< T1, T2, reversed >	1353
smrat::datastructures::OrderPair< T1, T2, true >	1353
smrat::cad::Origin This class represents one or more origins of some object	1354
smrat::cad::preprocessor::Origins	1356
smrat::OrPathShortener A rewriter that checks every (X)OR that only contains ANDs and UEQs as children for equalities that hold for every alternative, and adds such equalities explicitly	1357
Minisat::OutOfMemoryException	1358
smrat::parser::OutputWrapper	1358
Minisat::Map< K, D, H, E >::Pair	1358
smrat::pairhash< F, S, FirstHasher, SecondHasher > Note that this may NOT be a specialization of std::hash for std::pair, this is not allowed by the standard as it does not involve a user-defined type	1359
smrat::mcsat::ParallelExplanation< Backends >	1359
delta::Parser This class parses a whole smtlib file into a hierarchy of Node objects	1360
smrat::parser::ParserSettings	1361
smrat::parser::ParserState	1361
smrat::PBGaussModule< Settings >	1364
smrat::PBPPModule< Settings >	1391
smrat::PersistentUnionFind	1418
smrat::PFEModule< Settings >	1420
smrat::PModule	1446
smrat::cadcells::operators::properties::poly_additional_root_outside	1472
smrat::cadcells::operators::properties::poly_irreducible_semi_sgn_inv	1473
smrat::cadcells::operators::properties::poly_irreducible_sgn_inv	1474
smrat::cadcells::operators::properties::poly_ord_inv	1474
smrat::cadcells::operators::properties::poly_ord_inv_base	1475
smrat::cadcells::operators::properties::poly_pdel	1476
smrat::cadcells::operators::properties::poly_semi_sgn_inv	1476
smrat::cadcells::operators::properties::poly_sgn_inv	1477
smrat::cadcells::representation::util::PolyDelineation	1478
smrat::cadcells::representation::util::PolyDelineations	1478
smrat::cad::ProjectionPolynomialInformation::PolyInfo	1479
smrat::cad::PolynomialComparator< PolynomialGetter >	1479
smrat::cad::PolynomialLiftingQueue< PolynomialGetter >	1479
smrat::cadcells::datastructures::PolyPool A pool for polynomials	1481
smrat::cadcells::datastructures::detail::PolyProperties	1482
smrat::cadcells::datastructures::PolyRef Refers to a polynomial	1483
smrat::PolyTree	1483
smrat::PolyTreeContent	1484
smrat::PolyTreePool	1485
smrat::cad::Preprocessor	1486
smrat::cad::PreprocessorSettings	1487
benchmax::settings::PresetSettings	1488
smrat::PriorityQueue< T, Compare >	1488

<code>delta::Producer</code>	This class iteratively applies the operators to a smtlib file until no further simplifications can be performed	1490
<code>delta::ProgressBar</code>	This class provides a simple way to show progress bars on the command line	1491
<code>smtrat::cad::Projection< incrementality, backtracking, Settings ></code>		1492
<code>smtrat::qe::cad::Projection< Settings ></code>		1497
<code>smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings ></code>		1503
<code>smtrat::cad::Projection< Incrementality::FULL, BT, Settings ></code>		1509
<code>smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings ></code>		1509
<code>smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings ></code>	This class implements a projection that supports no incrementality and expects backtracking to be in order	1514
<code>smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings ></code>	This class implements a projection that supports no incrementality and allows backtracking to be out of order	1520
<code>smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings ></code>		1526
<code>smtrat::cad::projection_compare::ProjectionComparator< Strategy ></code>		1531
<code>smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::D ></code>		1531
<code>smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::LD ></code>		1532
<code>smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::ID ></code>		1532
<code>smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::PD ></code>		1532
<code>smtrat::cad::projection_compare::ProjectionComparator< ProjectionCompareStrategy::SD ></code>		1533
<code>smtrat::cad::projection_compare::ProjectionComparator_impl< Args ></code>		1533
<code>smtrat::cad::ProjectionGlobalInformation</code>		1533
<code>smtrat::cad::ProjectionInformation</code>		1535
<code>smtrat::cad::ProjectionLevelInformation</code>		1537
<code>smtrat::cad::ProjectionOperator</code>		1537
<code>smtrat::cad::ProjectionPolynomialInformation</code>		1538
<code>smtrat::cadcells::datastructures::Projections</code>	Encapsulates all computations on polynomials	1539
<code>smtrat::cadcells::operators::mccallum_filtered_impl::PropertiesSet</code>		1542
<code>smtrat::cadcells::operators::PropertiesSet< Op ></code>		1542
<code>smtrat::cadcells::operators::PropertiesSet< op::mccallum ></code>		1542
<code>smtrat::cadcells::datastructures::PropertiesT< Ts ></code>	Set of properties	1542
<code>smtrat::cadcells::datastructures::PropertiesT< T, Ts... ></code>		1543
<code>smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag ></code>		1543
<code>smtrat::cadcells::datastructures::PropertiesTContent< T, false ></code>		1543
<code>smtrat::cadcells::datastructures::PropertiesTContent< T, true ></code>		1543
<code>smtrat::cadcells::datastructures::property_hash< T ></code>		1544
<code>smtrat::PseudoBoolEncoder</code>	Base class for a PseudoBoolean Encoder	1544
<code>smtrat::PseudoBoolNormalizer</code>		1545
<code>smtrat::parser::QEParser</code>		1546
<code>smtrat::parser::QualifiedIdentifierParser</code>		1547
<code>smtrat::expression::QuantifierExpression</code>		1547
<code>smtrat::parser::QuantifierParser</code>		1548
<code>Minisat::Queue< T ></code>		1548
<code>smtrat::mcsat::icp::QueueEntry</code>		1549
<code>benchmax::RandomizationAdaptor< T ></code>	Provides iteration over a given std::vector in a pseudo-randomized order	1550
<code>smtrat::RationalCapsule</code>		1551
<code>smtrat::parser::RationalPolicies</code>	Specialization of qi::real_policies for a Rational	1552
<code>smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number ></code>	Represent a multidimensional point whose components are algebraic reals	1552
<code>smtrat::fixedsize_allocator< T >::rebind< U ></code>		1554
<code>smtrat::mcsat::onecellcad::recursive::RecursiveCAD</code>		1554

smrat::cad::projection::Reducta< Poly >	
Construct the set of reducta of the given polynomial	1558
Minisat::RegionAllocator< T >	1559
smrat::NNFRewriter::remove_xor_first_arg	1561
smrat::ReplaceVariablesRewriter	1561
delta::ErrorHandler::result< typename >	1563
smrat::parser::ErrorHandler::result< typename >	1563
smrat::cad::preprocessor::ResultantRule	1563
benchmax::Results	
Stores results for for whole benchmax run	1564
smrat::RNSEncoder	1565
smrat::cadcells::operators::properties::root_ordering_holds	1566
smrat::cadcells::operators::properties::root_well_def	1567
smrat::cadcells::datastructures::RootFunction	1568
smrat::mcsat::arithmetic::RootIndexer< RANT >	1570
smrat::cad::Sample	1570
smrat::cad::sample_compare::SampleComparator< Iterator, Strategy >	1573
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LS >	1573
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LT >	1573
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA >	1573
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS >	1574
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA >	1574
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::S >	1574
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::T >	1575
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA >	1575
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TS >	1575
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA >	1576
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >	1576
smrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Value >	1576
smrat::cad::sample_compare::SampleComparatorImpl< It, Args >	1577
smrat::cadcells::datastructures::SampledDerivation< Properties >	
A SampledDerivation is a DelineatedDerivation with a sample and an DelineationInterval w.r.t	1577
smrat::SampledDerivationRefCompare	1580
smrat::cad::SampleIteratorQueue< Iterator, Comparator >	1580
smrat::sampling< S >	1582
smrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING >	1582
smrat::SATModule< Settings >	
Implements a module performing DPLL style SAT checking	1582
smrat::parser::ScriptParser< Callee >	1614
smrat::parser::ParserState::ScriptScope	1616
smrat::mcsat::onecellcad::Section	
Represent a cell's (closed-interval-boundary) component along th k-th axis	1617
smrat::onecellcad::recursive::Section	
Represent a cell's closed-interval-boundary object along one single axis by an irreducible, multi-variate polynomial of level k	1617
smrat::mcsat::onecellcad::levelwise::SectionHeuristic1	1618
smrat::mcsat::onecellcad::levelwise::SectionHeuristic2	1618
smrat::mcsat::onecellcad::levelwise::SectionHeuristic3	1618
smrat::mcsat::onecellcad::Sector	
Represent a cell's open-interval boundary object along one single axis by two irreducible, multi-variate polynomials of level k	1619
smrat::onecellcad::recursive::Sector	
Represent a cell's open-interval boundary object along one single axis by two irreducible, multi-variate polynomials of level k	1619
smrat::mcsat::onecellcad::levelwise::SectorHeuristic1	1620
smrat::mcsat::onecellcad::levelwise::SectorHeuristic2	1620
smrat::mcsat::onecellcad::levelwise::SectorHeuristic3	1621

smrat::subtropical::Separator	
Represents the class of all original constraints with the same left hand side after a normalization	1621
smrat::mcsat::SequentialAssignment< Backends >
smrat::mcsat::SequentialExplanation< Backends >
benchmax::settings::Settings	
Generic class to manage runtime settings
delta::Settings	
This class loads and checks the command line options with help of boost::program_<options>
smrat::settings::Settings
smrat::SettingsComponents
benchmax::SettingsParser	
Generic class to manage settings parsing using boost::program_options
smrat::SettingsParser
smrat::CNFerModule::SettingsType
smrat::parser::SExpressionParser
smrat::parser::SExpressionSequence< T >
smrat::ShortFormulaEncoder
benchmax::simple_parser
smrat::parser::Theories::SimpleSortAdder	
Helper functor for addSimpleSorts() method
smrat::parser::SimpleSymbolParser	
Parses symbols: simple_symbol quoted_symbol where
smrat::expression::simplifier::Simplifier
smrat::expression::simplifier::SimplifierChainCaller< chainID >
smrat::expression::simplifier::SimplifierChainCaller< 0 >
smrat::expression::simplifier::SingletonSimplifier
smrat::cad::sample_compare::size
delta::Skipper	
This class is a boost::spirit::qi grammar that matches whitespaces and smtlib comments
smrat::parser::Skipper
benchmax::SlurmBackend	
Backend for the Slurm workload manager
benchmax::settings::SlurmBackendSettings	
Settings for the Slurm backend
smrat::parser::SMTLIBParser
benchmax::SMTRAT	
Tool wrapper for SMT-RAT for SMT-LIB
benchmax::SMTRAT_Analyzer	
Tool wrapper for SMT-RAT for SMT-LIB
benchmax::SMTRAT_OPB	
Adapts the SMTRAT solver for OPB files
smrat::execution::SoftAssertion
smrat::settings::SolverSettings
smrat::parser::SortedVariableParser
smrat::parser::SortParser
smrat::parser::SpecConstantParser
smrat::SplitSOSModule< Settings >
benchmax::SSHBackend
benchmax::settings::SSHBackendSettings	
Settings for SSH backend
benchmax::ssh::SSHConnection	
A wrapper class that manages a single SSH connection as specified in a Node object (with all its channels)
smrat::vs::State
smrat::StaticUnionFind
smrat::statistics::StatisticsSettings

smtrat::StrategyGraph	1711
delta::String	
This class can be used to create <code>std::string</code> objects using streaming operators	1712
Minisat::StringOption	1713
smtrat::parser::StringParser	
Parses strings: ".+" with escape sequences "\\" and "\\\\"	1714
smtrat::STropModule< Settings >	1715
benchmax::slurm::SubmitfileProperties	
All properties needed to create a submit file	1742
smtrat::vs::Substitution	1743
smtrat::vs::substitutionPointerEqual	1747
smtrat::vs::substitutionPointerHash	1747
smtrat::cadcells::datastructures::SymbolicInterval	
A symbolic interval whose bounds are defined by indexed root expressions	1748
delta::SymbolParser	
This class is a <code>boost::spirit::qi</code> grammar that matches all kinds of smtlib symbols, numbers etc	1749
smtrat::parser::SymbolParser	1750
smtrat::SymmetryModule< Settings >	1750
smtrat::lra::Tableau< Settings, T1, T2 >	1778
smtrat::lra::TableauEntry< T1, T2 >	1796
smtrat::cadcells::datastructures::TaggedIndexedRoot	1799
smtrat::mcsat::onecellcad::TagPoly	
Tagged Polynomials	1799
smtrat::Task	1800
delta::TempFilenameGenerator	
This class generates and reuses temporary filenames with a common prefix	1801
smtrat::parser::TermParser	1802
smtrat::mcsat::vs::helper::TestCandidate	1803
smtrat::parser::Theories	
Combines all implemented theories and provides a single interface to interact with all theories at once	1804
smtrat::parser::TheoryError	1807
smtrat::mcsat::TheoryLevel	1807
smtrat::Module::TheoryPropagation	1808
smtrat::TheoryVarSchedulerStatic< vot >	
Schedules theory variables statically	1809
smtrat::ThreadPool	1813
smtrat::cad::debug::TikzBasePrinter	1813
smtrat::cad::debug::TikzDAGPrinter	1815
smtrat::cad::debug::TikzHistoryPrinter	1816
smtrat::cad::debug::TikzTreePrinter	1817
benchmax::Tool	
Base class for any tool	1819
benchmax::settings::ToolSettings	
Tool-related settings	1822
smtrat::TotalizerEncoder	
TotalizerEncoder implements the Totalizer encoding as described in "Incremental Cardinality Constraints for MaxSAT" by Martins et al https://doi.org/10.1007/978-3-319-10428-7_39	1823
smtrat::TotalizerTree	1825
smtrat::cad::projection_compare::type	1826
smtrat::cad::sample_compare::type	1826
smtrat::uf_impl::uf_data_wrapper< T, compression >	
Wrapper around data entry and <code>uf_rank_and_class</code> ; this class contains a value of type T and both rank and class of an entry in the UF datastructure	1826
smtrat::uf_impl::uf_data_wrapper< void, compression >	
Special case of the <code>uf_data_wrapper</code> in case the data type is void	1827

<code>smtrat::uf_impl::uf_rank_and_class< compression ></code>	Wrapper class that contains the rank and parent class of some entry of the UF datastructure	1828
<code>smtrat::uf_impl::uf_rank_and_class< true ></code>	Compressed implementation of <code>uf_rank_and_class</code>	1829
<code>smtrat::UFCEgarModule< Settings ></code>		1830
<code>smtrat::UFRewriter</code>		1856
<code>smtrat::expression::UnaryExpression</code>		1857
<code>smtrat::cad::debug::UnifiedData</code>		1857
<code>smtrat::parser::types::UninterpretedTheory</code>	Types of the theory of equalities and uninterpreted functions	1858
<code>smtrat::parser::UninterpretedTheory</code>	Implements the theory of equalities and uninterpreted functions	1859
<code>smtrat::union_find< T, no_compression, MaxTracebackLength ></code>	Implements a union-find datastructure, a datastructure containing an array of entries	1862
<code>smtrat::UnionFindInterface< T, Implementation ></code>		1866
<code>smtrat::UnionFindModule< Settings ></code>		1867
<code>smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher</code>		1894
<code>smtrat::UnsatCore< Solver, Strategy ></code>		1894
<code>smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy ></code>		1895
<code>smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion ></code>	Implements an easy strategy to obtain an unsatisfiable core	1895
<code>smtrat::vs::unsignedTripleCmp</code>		1896
<code>smtrat::parser::UserFunctionInstantiator</code>		1897
<code>smtrat::validation::ValidationCollector</code>		1898
<code>smtrat::validation::ValidationPoint</code>		1898
<code>smtrat::validation::ValidationPrinter< SOF ></code>		1900
<code>smtrat::validation::ValidationSettings</code>		1900
<code>smtrat::lra::Value< T ></code>		1900
<code>smtrat::lra::Variable< T1, T2 ></code>		1908
<code>smtrat::vb::Variable< T ></code>	Class for a variable	1918
<code>smtrat::vb::VariableBounds< T ></code>	Class to manage the bounds of a variable	1921
<code>smtrat::VariableCapsule</code>		1928
<code>smtrat::mcsat::variableordering::VariableIDs</code>		1929
<code>smtrat::VariableRewriteRule</code>		1929
<code>smtrat::variant_hash_visitor</code>		1930
<code>smtrat::parser::conversion::VariantConverter< Res ></code>	Converts variants to some type using the <code>Converter</code> class	1931
<code>smtrat::VariantMap< Key, Value ></code>	This class is a specialization of <code>std::map</code> where the keys are of arbitrary type and the values are of type <code>boost::variant</code>	1932
<code>smtrat::parser::conversion::VariantVariantConverter< Res ></code>	Converts variants to another variant type not using the <code>Converter</code> class	1934
<code>smtrat::VarSchedulerBase</code>	Base class for variable schedulers	1935
<code>smtrat::VarSchedulerFixedRandom</code>		1938
<code>smtrat::VarSchedulerMcsatActivityPreferTheory< vot ></code>	Activity-based scheduler preferring initially theory variables	1941
<code>smtrat::VarSchedulerMcsatBase</code>	Base class for all MCSAT variable scheduler	1945
<code>smtrat::VarSchedulerMcsatBooleanFirst< vot ></code>	Variable scheduling that all decides Boolean variables first before deciding any theory variable	1948
<code>smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler ></code>	Variable scheduling that all decides theory variables first before deciding any Boolean variable	1952
<code>smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities ></code>	Decides only Constraints occurring in clauses that are univariate in the current theory variable while the theory ordering is static	1955

<code>smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot ></code>	
Decides only Constraints univariate in the current theory variable while the theory ordering is static	1959
<code>smtrat::VarSchedulerMinisat</code>	
Minisat's activity-based variable scheduling	1963
<code>smtrat::VarSchedulerRandom</code>	
Random scheduler	1966
<code>smtrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic ></code>	
Scheduler for SMT, implementing theory guided heuristics	1969
<code>Minisat::vec< T ></code>	
.	1972
<code>smtrat::parser::conversion::VectorVariantConverter< Res ></code>	
Converts a vector of variants to a vector of some type using the Converter class	1975
<code>smtrat::subtropical::Vertex</code>	
Represents a term of an original constraint and assigns him a rating variable if a weak separator is searched	1976
<code>smtrat::VSMModule< Settings ></code>	
.	1977
<code>Minisat::Watcher</code>	
[Minisat related code]	2005
<code>smtrat::ICPModule< Settings >::weights</code>	
.	2006
<code>benchmax::XMLWriter</code>	
Writes results to a xml file	2006
<code>benchmax::Z3</code>	
Tool wrapper for <code>Z3</code> for SMT-LIB	2007

0.14 Namespace Documentation

0.14.1 benchmax Namespace Reference

Namespaces

- `settings`
- `slurm`
- `ssh`

Data Structures

- class `Backend`

Base class for all backends.
- class `CondorBackend`

Backend for the HTCondor batch system.
- class `RandomizationAdaptor`

Provides iteration over a given std::vector in a pseudo-randomized order.
- class `Jobs`

Represents a set of jobs, constructed as the cartesian product of a set of tools and a set of benchmarks.
- class `LocalBackend`

This backend simply runs files sequentially on the local machine.
- class `SlurmBackend`

Backend for the Slurm workload manager.
- class `SSHBackend`
- class `BenchmarkSet`

A set of benchmarks from some common base directory.
- struct `BenchmarkResult`

Results for a single benchmark run.
- class `Database`

Dummy database that effectively disables storing to database. Set BENCHMAX_DATABASE to actually use a database.

- class [DBAL](#)
- class [Results](#)
Stores results for for whole benchmax run.
- class [XMLWriter](#)
Writes results to a xml file.
- class [SettingsParser](#)
Generic class to manage settings parsing using boost::program_options.
- class [MathSAT](#)
Tool wrapper for [MathSAT](#) for SMT-LIB.
- class [Minisat](#)
Tool wrapper for a [Minisat](#) solver.
- class [Minisatp](#)
Tool wrapper for the [Minisatp](#) solver for pseudo-Boolean problems.
- class [simple_parser](#)
- class [SMTRAT](#)
Tool wrapper for SMT-RAT for SMT-LIB.
- class [SMTRAT_Analyzer](#)
Tool wrapper for SMT-RAT for SMT-LIB.
- class [SMTRAT_OPB](#)
Adapts the [SMTRAT](#) solver for OPB files.
- class [Tool](#)
Base class for any tool.
- class [Z3](#)
Tool wrapper for [Z3](#) for SMT-LIB.

TypeDefs

- using [Job](#) = std::pair< const [Tool](#) *, std::filesystem::path >
TypeDef for a job, consisting of a tool and an input file.
- using [ToolPtr](#) = std::unique_ptr< [Tool](#) >
A std::unique_ptr to a [Tool](#).
- using [Tools](#) = std::vector< [ToolPtr](#) >
A vector of ToolPtr.

Functions

- template<typename Backend >
void [execute_backend](#) (const std::string &name, const [Jobs](#) &jobs)
- void [run_backend](#) (const std::string &backend, const [Jobs](#) &jobs)
Runs a given backend on a list of tools and benchmarks.
- const auto & [settings_slurm](#) ()
Return the Slurm settings.
- const auto & [settings_ssh](#) ()
Return the SSH settings.
- [BenchmarkSet](#) [loadBenchmarks](#) ()
Loads benchmark files from directory specified in settings.
- const auto & [settings_benchmarks](#) ()
Return the benchmark settings.
- void [logging_configure](#) ()
Configure default logging for benchmax.
- void [logging_quiet](#) ()
Configure quiet logging for benchmax.

- void `logging_verbose ()`
Configure verbose logging for benchmax.
- std::ostream & `operator<< (std::ostream &os, const BenchmarkResult &results)`
Streaming operator for `BenchmarkResult`.
- const auto & `settings_preset ()`
Return the Slurm settings.
- template<typename T>
const auto & `settings_get (const std::string &name)`
Helper function to obtain settings by name and type.
- const auto & `settings_core ()`
Retrieved core settings.
- const auto & `settings_operation ()`
Retrieved operation settings.
- bool `is_system_lib (std::string path)`
- std::ostream & `operator<< (std::ostream &os, const Tool &tool)`
Streaming operator for a generic tool.
- template<typename T>
void `createTools (const std::vector< std::filesystem::path > &arguments, Tools &tools)`
Create tools of a given type `T` from a list of binaries and store them in `tools`.
- `Tools loadTools ()`
Load all tools from the tool settings.
- const auto & `settings_tools ()`
Returns the tool settings.
- int `call_program (const std::string &commandline, std::string &stdout, bool print_to_stdout=false)`
Runs an external program from some command line and records the output to `stdout`.
- std::filesystem::path `common_prefix (const std::filesystem::path &p1, const std::filesystem::path &p2)`
Computes the common prefix of two paths.
- std::filesystem::path `common_prefix (const std::vector< std::filesystem::path > &s, bool skip_last=true)`
Computes the common prefix of a list of paths.
- std::filesystem::path `common_prefix (const std::initializer_list< std::vector< std::filesystem::path > >&s, bool skip_last=true)`
Computes the common prefix of multiple lists of paths.
- std::filesystem::path `remove_prefix (const std::filesystem::path &s, const std::filesystem::path &prefix)`
Remove a prefix from a path.
- bool `is_extension (const std::filesystem::path &path, const std::string &extension)`
Checks whether the extension of the filename is as specified.
- std::size_t `parse_peak_memory (const std::string &output)`

0.14.1.1 Typedef Documentation

0.14.1.1.1 Job using `benchmax::Job = decltype (std::pair<const Tool*, std::filesystem::path>)`
Typedef for a job, consisting of a tool and an input file.

0.14.1.1.2 ToolPtr using `benchmax::ToolPtr = std::unique_ptr<Tool>`
A `std::unique_ptr` to a `Tool`.

0.14.1.1.3 Tools using `benchmax::Tools = std::vector<ToolPtr>`
A vector of `ToolPtr`.

0.14.1.2 Function Documentation

```
0.14.1.2.1 call_program() int benchmax::call_program (
    const std::string & commandline,
    std::string & stdout,
    bool print_to_stdout = false ) [inline]
```

Runs an external program from some command line and records the output to stdout.
Prints the program output and records it at the same time if *print_to_stdout* is set to true.

Parameters

<i>commandline</i>	Program to execute.
<i>stdout</i>	Standard output.
<i>print_to_stdout</i>	Also print if true.

Returns

Exit code of the program.

```
0.14.1.2.2 common_prefix() [1/3] std::filesystem::path benchmax::common_prefix (
    const std::filesystem::path & p1,
    const std::filesystem::path & p2 ) [inline]
```

Computes the common prefix of two paths.

```
0.14.1.2.3 common_prefix() [2/3] std::filesystem::path benchmax::common_prefix (
    const std::initializer_list< std::vector< std::filesystem::path >> & s,
    bool skip_last = true ) [inline]
```

Computes the common prefix of multiple lists of paths.

skip_last specifies whether the last part if the path (usually the filename) may be part of the prefix.

```
0.14.1.2.4 common_prefix() [3/3] std::filesystem::path benchmax::common_prefix (
    const std::vector< std::filesystem::path > & s,
    bool skip_last = true ) [inline]
```

Computes the common prefix of a list of paths.

skip_last specifies whether the last part if the path (usually the filename) may be part of the prefix.

```
0.14.1.2.5 createTools() template<typename T >
void benchmax::createTools (
    const std::vector< std::filesystem::path > & arguments,
    Tools & tools )
```

Create tools of a given type T from a list of binaries and store them in tools.

```
0.14.1.2.6 execute_backend() template<typename Backend >
void benchmax::execute_backend (
    const std::string & name,
    const Jobs & jobs )
```

0.14.1.2.7 `is_extension()` `bool benchmax::is_extension (`
 `const std::filesystem::path & path,`
 `const std::string & extension) [inline]`

Checks whether the extension of the filename is as specified.

0.14.1.2.8 `is_system_lib()` `bool benchmax::is_system_lib (`
 `std::string path) [inline]`

0.14.1.2.9 `loadBenchmarks()` `BenchmarkSet benchmax::loadBenchmarks ()`
Loads benchmark files from directory specified in settings.

0.14.1.2.10 `loadTools()` `Tools benchmax::loadTools ()`
Load all tools from the tool settings.

0.14.1.2.11 `logging_configure()` `void benchmax::logging_configure () [inline]`
Configure default logging for benchmax.

0.14.1.2.12 `logging_quiet()` `void benchmax::logging_quiet () [inline]`
Configure quiet logging for benchmax.

0.14.1.2.13 `logging_verbose()` `void benchmax::logging_verbose () [inline]`
Configure verbose logging for benchmax.

0.14.1.2.14 `operator<<()` [1/2] `std::ostream& benchmax::operator<< (`
 `std::ostream & os,`
 `const BenchmarkResult & results) [inline]`

Streaming operator for `BenchmarkResult`.

0.14.1.2.15 `operator<<()` [2/2] `std::ostream& benchmax::operator<< (`
 `std::ostream & os,`
 `const Tool & tool) [inline]`

Streaming operator for a generic tool.

0.14.1.2.16 `parse_peak_memory()` `std::size_t benchmax::parse_peak_memory (`
 `const std::string & output) [inline]`

0.14.1.2.17 `remove_prefix()` `std::filesystem::path benchmax::remove_prefix (`
 `const std::filesystem::path & s,`
 `const std::filesystem::path & prefix) [inline]`

Remove a prefix from a path.

```
0.14.1.2.18 run_backend() void benchmax::run_backend (
    const std::string & backend,
    const Jobs & jobs )
```

Runs a given backend on a list of tools and benchmarks.

Parameters

<i>backend</i>	Backend name.
<i>tools</i>	List of tools.
<i>benchmarks</i>	List of benchmarks.

0.14.1.2.19 settings_benchmarks() const auto& benchmax::settings_benchmarks () [inline]
Return the benchmark settings.

0.14.1.2.20 settings_core() const auto& benchmax::settings_core () [inline]
Retrieved core settings.

0.14.1.2.21 settings_get() template<typename T >
const auto& benchmax::settings_get (
 const std::string & name) [inline]
Helper function to obtain settings by name and type.

0.14.1.2.22 settings_operation() const auto& benchmax::settings_operation () [inline]
Retrieved operation settings.

0.14.1.2.23 settings_preset() const auto& benchmax::settings_preset () [inline]
Return the Slurm settings.

0.14.1.2.24 settings_slurm() const auto& benchmax::settings_slurm () [inline]
Return the Slurm settings.

0.14.1.2.25 settings_ssh() const auto& benchmax::settings_ssh () [inline]
Return the SSH settings.

0.14.1.2.26 settings_tools() const auto& benchmax::settings_tools () [inline]
Returns the tool settings.

0.14.2 benchmax::settings Namespace Reference

Data Structures

- struct [SlurmBackendSettings](#)
Settings for the Slurm backend.
- struct [SSHBackendSettings](#)
Settings for SSH backend.
- struct [BenchmarkSettings](#)
Settings for benchmarks.
- struct [PresetSettings](#)
- struct [CoreSettings](#)
Core settings.
- struct [OperationSettings](#)

- struct **Settings**
Generic class to manage runtime settings.
- struct **ToolSettings**
Tool-related settings.

Functions

- void **registerSlurmBackendSettings** (**SettingsParser** *parser)
Registers Slurm settings with the settings parser.
- void **registerSSHBackendSettings** (**SettingsParser** *parser)
Registers SSH settings with settings parser.
- bool **finalize_benchmark_settings** (**BenchmarkSettings** &s, const **boost::program_options::variables_map** &)
Postprocess benchmark settings.
- void **registerBenchmarkSettings** (**SettingsParser** *parser)
Registers benchmark settings with the settings parser.
- bool **finalize_preset_settings** (**PresetSettings** &s, **boost::program_options::variables_map** &values)
Postprocess preset settings.
- void **registerPresetSettings** (**SettingsParser** *parser)
Registers preset settings with the settings parser.
- bool **finalize_tool_settings** (**ToolSettings** &s, const **boost::program_options::variables_map** &)
Postprocess settings to compute common prefix.
- void **registerToolSettings** (**SettingsParser** *parser)
Registers tool settings with the settings parser.

0.14.2.1 Function Documentation

0.14.2.1.1 finalize_benchmark_settings() `bool benchmax::settings::finalize_benchmark_settings (`
`BenchmarkSettings & s,`
`const boost::program_options::variables_map &)`

Postprocess benchmark settings.

0.14.2.1.2 finalize_preset_settings() `bool benchmax::settings::finalize_preset_settings (`
`PresetSettings & s,`
`boost::program_options::variables_map & values)`

Postprocess preset settings.

0.14.2.1.3 finalize_tool_settings() `bool benchmax::settings::finalize_tool_settings (`
`ToolSettings & s,`
`const boost::program_options::variables_map &)`

Postprocess settings to compute common prefix.

0.14.2.1.4 registerBenchmarkSettings() `void benchmax::settings::registerBenchmarkSettings (`
`SettingsParser * parser)`

Registers benchmark settings with the settings parser.

0.14.2.1.5 registerPresetSettings() `void benchmax::settings::registerPresetSettings (`
 `SettingsParser * parser)`

Registers preset settings with the settings parser.

0.14.2.1.6 registerSlurmBackendSettings() `void benchmax::settings::registerSlurmBackendSettings (`
 `SettingsParser * parser)`

Registers Slurm settings with the settings parser.

0.14.2.1.7 registerSSHBackendSettings() `void benchmax::settings::registerSSHBackendSettings (`
 `SettingsParser * parser)`

Registers SSH settings with settings parser.

0.14.2.1.8 registerToolSettings() `void benchmax::settings::registerToolSettings (`
 `SettingsParser * parser)`

Registers tool settings with the settings parser.

0.14.3 benchmax::slurm Namespace Reference

Data Structures

- struct [ArchiveProperties](#)
All properties needed to archive log files.
- struct [SubmitfileProperties](#)
All properties needed to create a submit file.
- struct [ChunkedSubmitfileProperties](#)
All properties needed to create a submit file.

Functions

- void [archive_log_files](#) (const [ArchiveProperties](#) &p)
Put all log files into an archive.
- std::vector< fs::path > [collect_result_files](#) (const fs::path &basedir)
Collects all result files in the given base directory for this job id.
- std::string [generate_submit_file](#) (const [SubmitfileProperties](#) &p)
Generate a submit file for Slurm with the given properties.
- std::string [generate_submit_file_chunked](#) (const [ChunkedSubmitfileProperties](#) &p)
- int [parse_job_id](#) (const std::string &output)
Parses the job id from the output of sbatch.
- std::string [parse_result_info](#) (const std::string &content, const std::string &name)
Parse a single result information from the output.
- void [remove_log_files](#) (const std::vector< fs::path > &files, bool remove)
Remove the given list of files.
- void [clear_log_files](#) (const fs::path &basedir)
Clear log files from directory.
- bool [is_job_finished](#) (int jobid)
Checks if the given job is finished.
- template<typename Jobs >
void [generate_jobs_file](#) (const std::string &filename, std::pair< std::size_t, std::size_t > range, const [Jobs](#) &jobs)

0.14.3.1 Function Documentation

0.14.3.1.1 archive_log_files() `void benchmax::slurm::archive_log_files (`
 `const ArchiveProperties & p)`

Put all log files into an archive.

0.14.3.1.2 clear_log_files() `void benchmax::slurm::clear_log_files (`
 `const fs::path & basedir)`

Clear log files from directory.

Parameters

<code>basedir</code>	Base directory to search in.
----------------------	------------------------------

0.14.3.1.3 collect_result_files() `std::vector< fs::path > benchmax::slurm::collect_result_files (`
 `const fs::path & basedir)`

Collects all result files in the given base directory for this job id.

Parameters

<code>basedir</code>	Base directory to search in.
<code>jobid</code>	ID of slurm job.

Returns

List of result files.

0.14.3.1.4 generate_jobs_file() `template<typename Jobs >`
`void benchmax::slurm::generate_jobs_file (`
 `const std::string & filename,`
 `std::pair< std::size_t, std::size_t > range,`
 `const Jobs & jobs)`

0.14.3.1.5 generate_submit_file() `std::string benchmax::slurm::generate_submit_file (`
 `const SubmitfileProperties & p)`

Generate a submit file for Slurm with the given properties.

Parameters

<code>p</code>	Collection of all information needed.
----------------	---------------------------------------

Returns

The filename of the submit file.

0.14.3.1.6 generate_submit_file_chunked() `std::string benchmax::slurm::generate_submit_file_`
 `chunked (`

```
const ChunkedSubmitfileProperties & p )
```

0.14.3.1.7 `is_job_finished()` `bool benchmax::slurm::is_job_finished (int jobid)`

Checks if the given job is finished.

Parameters

<code>jobid</code>	The job id.
--------------------	-------------

Returns

True if the job is completed.

0.14.3.1.8 `parse_job_id()` `int benchmax::slurm::parse_job_id (const std::string & output)`

Parses the job id from the output of sbatch.

Parameters

<code>output</code>	Output of sbatch.
---------------------	-------------------

Returns

Slurm job id.

0.14.3.1.9 `parse_result_info()` `std::string benchmax::slurm::parse_result_info (const std::string & content, const std::string & name)`

Parse a single result information from the output.

Parameters

<code>content</code>	Result information output.
<code>name</code>	Information to be parsed.

Returns

Information specified in the output.

0.14.3.1.10 `remove_log_files()` `void benchmax::slurm::remove_log_files (const std::vector< fs::path > & files, bool remove)`

Remove the given list of files.

Parameters

<code>files</code>	List of files.
<code>remove</code>	Boolean flag whether to actually remove files.

0.14.4 `benchmax::ssh` Namespace Reference

Data Structures

- struct `Node`

Specification of a computation node for the SSH backend.
- class `SSHConnection`

A wrapper class that manages a single SSH connection as specified in a `Node` object (with all its channels).

0.14.5 `carl` Namespace Reference

0.14.6 `delta` Namespace Reference

Data Structures

- class `Checker`

This class takes care of calling the actual solver.
- class `Consumer`

This class takes care of asynchronous execution of calls to the solver.
- struct `Node`

This class represents a node in a SMTLIB file.
- struct `NodePrinter`
- struct `Skipper`

This class is a `boost::spirit::qi` grammar that matches whitespaces and smtlib comments.
- struct `SymbolParser`

This class is a `boost::spirit::qi` grammar that matches all kinds of smtlib symbols, numbers etc.
- struct `ErrorHandler`
- class `Parser`

This class parses a whole smtlib file into a hierarchy of `Node` objects.
- class `Producer`

This class iteratively applies the operators to a smtlib file until no further simplifications can be performed.
- class `Settings`

This class loads and checks the command line options with help of `boost::program_options`.
- class `TempFilenameGenerator`

This class generates and reuses temporary filenames with a common prefix.
- class `ProgressBar`

This class provides a simple way to show progress bars on the command line.
- class `String`

This class can be used to create `std::string` objects using streaming operators.

TypeDefs

- using `NodeChangeSet` = `std::vector< Node >`

Set of new nodes.
- using `NodeOperator` = `std::function< NodeChangeSet(const Node &)>`

Type of an node operator.
- typedef `boost::spirit::istream_iterator` `BaseliteratorType`
- typedef `boost::spirit::line_pos_iterator< BaseliteratorType >` `PositionIteratorType`
- typedef `PositionIteratorType` `Iterator`

Functions

- `std::ostream & operator<< (std::ostream &os, const Node &n)`
Streaming operator.
- `template<bool pretty>`
`std::ostream & operator<< (std::ostream &os, const NodePrinter< pretty > &np)`
- `NodeChangeSet children (const Node &n)`
Node operator that returns all children of a node.
- `NodeChangeSet reorderChildren (const Node &n)`
Node operator that reorders children of a node.
- `NodeChangeSet mergeChild (const Node &n)`
Node operator that merges nodes with one of its own children.
- `NodeChangeSet number (const Node &n)`
Node operator that provides meaningful replacements for numbers.
- `NodeChangeSet constant (const Node &n)`
Node operator that provides meaningful replacements for variables.
- `NodeChangeSet letExpression (const Node &n)`
Node operator that eliminated let expressions.
- `NodeChangeSet BV_zeroExtend (const Node &n)`
- `NodeChangeSet BV_mergeShift (const Node &n)`

$$(bvlshr (bvlshr x (_ bv1 8)) (_ bv1 8))$$

0.14.6.1 Typedef Documentation

0.14.6.1.1 BaselIteratorType `typedef boost::spirit::istream_iterator delta::BaseIteratorType`

0.14.6.1.2 Iterator `typedef PositionIteratorType delta::Iterator`

0.14.6.1.3 NodeChangeSet `using delta::NodeChangeSet = typedef std::vector<Node>`
Set of new nodes.

0.14.6.1.4 NodeOperator `using delta::NodeOperator = typedef std::function<NodeChangeSet (const Node &) >`

Type of an node operator.

A NodeOperator is called on a node and shall return a set of new nodes. These new nodes are supposed to be a simplifying replacement for the given node.

0.14.6.1.5 PositionIteratorType `typedef boost::spirit::line_pos_iterator<BaseIteratorType> delta::PositionIterator`

0.14.6.2 Function Documentation

0.14.6.2.1 BV_mergeShift() `NodeChangeSet delta::BV_mergeShift (`
`const Node & n)`
`(bvlshr (bvlshr x (_ bv1 8)) (_ bv1 8))`

0.14.6.2.2 `BV_zeroExtend()` `NodeChangeSet` `delta::BV_zeroExtend (`
`const Node & n)`

0.14.6.2.3 `children()` `NodeChangeSet` `delta::children (`
`const Node & n)`

`Node` operator that returns all children of a node.

Parameters

<code>n</code>	<code>Node.</code>
----------------	--------------------

Returns

A set of replacements.

0.14.6.2.4 `constant()` `NodeChangeSet` `delta::constant (`
`const Node & n)`

`Node` operator that provides meaningful replacements for variables.

Parameters

<code>n</code>	<code>Node.</code>
----------------	--------------------

Returns

A set of replacements.

0.14.6.2.5 `letExpression()` `NodeChangeSet` `delta::letExpression (`
`const Node & n)`

`Node` operator that eliminated let expressions.

Parameters

<code>n</code>	<code>Node.</code>
----------------	--------------------

Returns

A set of replacements.

0.14.6.2.6 `mergeChild()` `NodeChangeSet` `delta::mergeChild (`
`const Node & n)`

`Node` operator that merges nodes with one of its own children.

Parameters

<code>n</code>	<code>Node.</code>
----------------	--------------------

Returns

A set of replacements.

0.14.6.2.7 number() `NodeChangeSet` `delta::number (`
 `const Node & n)`

`Node` operator that provides meaningful replacements for numbers.

Parameters

<code>n</code>	<code>Node.</code>
----------------	--------------------

Returns

A set of replacements.

0.14.6.2.8 operator<<() [1/2] `std::ostream&` `delta::operator<< (`
 `std::ostream & os,`
 `const Node & n)` [inline]

Streaming operator.

This operator should output a representation that is syntactically identically to the one that was parsed.

Parameters

<code>os</code>	Output stream.
<code>n</code>	<code>Node.</code>

Returns

`os.`

0.14.6.2.9 operator<<() [2/2] `template<bool pretty>`
`std::ostream&` `delta::operator<< (`
 `std::ostream & os,`
 `const NodePrinter< pretty > & np)` [inline]

0.14.6.2.10 reorderChildren() `NodeChangeSet` `delta::reorderChildren (`
 `const Node & n)`

`Node` operator that reorders children of a node.

Parameters

<code>n</code>	<code>Node.</code>
----------------	--------------------

Returns

A set of replacements.

0.14.7 Minisat Namespace Reference

Data Structures

- class [RegionAllocator](#)
- class [Heap](#)
- struct [Hash](#)
- struct [Equal](#)
- struct [DeepHash](#)
- struct [DeepEqual](#)
- class [Map](#)
- class [Option](#)
- struct [IntRange](#)
- struct [Int64Range](#)
- struct [DoubleRange](#)
- class [DoubleOption](#)
- class [IntOption](#)
- class [StringOption](#)
- class [BoolOption](#)
- class [Queue](#)
- struct [Lit](#)
- class [Ibool](#)
- class [Clause](#)
- class [ClauseAllocator](#)
- class [OccLists](#)
- class [CMap](#)
- struct [Watcher](#)
 - [*Minisat related code*]
- struct [LessThan_default](#)
- class [vec](#)
- class [OutOfMemoryException](#)

Typedefs

- typedef int [Var](#)
- typedef [RegionAllocator](#)< uint32_t >::Ref [CRef](#)

Functions

- template<class V , class T >
static void [remove](#) (V &ts, const T &t)
- template<class V , class T >
static bool [find](#) (V &ts, const T &t)
- template<class T >
static void [copy](#) (const T &from, T &to)
- template<class T >
static void [copy](#) (const [vec](#)< T > &from, [vec](#)< T > &to, bool [append](#)=false)
- template<class T >
static void [append](#) (const [vec](#)< T > &from, [vec](#)< T > &to)
- static uint32_t [hash](#) (uint32_t x)
- static uint32_t [hash](#) (uint64_t x)
- static uint32_t [hash](#) (int32_t x)
- static uint32_t [hash](#) (int64_t x)

- void `printUsageAndExit` (int argc, char **argv, bool verbose=false)
- void `setUsageHelp` (const char *str)
- void `setHelpPrefixStr` (const char *str)
- `Lit mkLit` (Var var, bool sign=false)
- `Lit operator~` (Lit p)
- `Lit operator^` (Lit p, bool b)
- bool `sign` (Lit p)
- int `var` (Lit p)
- `Lit neg` (Lit p)
- int `toInt` (Var v)
- int `toInt` (Lit p)
- `Lit toLit` (int i)
- std::ostream & `operator<<` (std::ostream &os, const Lit &l)
- int `toInt` (lbool l)
- `lbool toLbool` (int v)
- std::ostream & `operator<<` (std::ostream &os, Minisat::lbool b)
- std::ostream & `operator<<` (std::ostream &os, const Minisat::Clause &c)
- template<class T, class LessThan >
void `selectionSort` (T *array, int size, LessThan lt)
- template<class T >
static void `selectionSort` (T *array, int size)
- template<class T, class LessThan >
void `sort` (T *array, int size, LessThan lt)
- template<class T >
static void `sort` (T *array, int size)
- template<class T, class LessThan >
void `sort` (vec< T > &v, LessThan lt)
- template<class T >
void `sort` (vec< T > &v)
- template<typename T >
std::ostream & `operator<<` (std::ostream &os, const vec< T > &v)
- template<> std::ostream & `operator<<` (std::ostream &os, const vec< char > &v)
- static void * `xrealloc` (void *ptr, size_t size)

Variables

- static const int `nprimes` = 25
- static const int `primes` [`nprimes`] = { 31, 73, 151, 313, 643, 1291, 2593, 5233, 10501, 21013, 42073, 84181, 168451, 337219, 674701, 1349473, 2699299, 5398891, 10798093, 21596719, 43193641, 86387383, 172775299, 345550609, 691101253 }
- const `Lit lit_Undef` = { -2 }
- const `Lit lit_Error` = { -1 }
- static const unsigned `NORMAL_CLAUSE` = 0
- static const unsigned `LEMMA_CLAUSE` = 1
- static const unsigned `CONFLICT_CLAUSE` = 2
- static const unsigned `PERMANENT_CLAUSE` = 3
- const `CRef CRef_Undef` = RegionAllocator<uint32_t>::Ref_Undef
- const `CRef CRef_Lazy` = RegionAllocator<uint32_t>::Ref_Undef - 1
- const `CRef CRef_TPropagation` = RegionAllocator<uint32_t>::Ref_Undef - 2

0.14.7.1 Typedef Documentation

0.14.7.1.1 `CRef` `typedef RegionAllocator<uint32_t>::Ref Minisat::CRef`

0.14.7.1.2 Var `typedef int Minisat::Var`

0.14.7.2 Function Documentation

0.14.7.2.1 append() `template<class T >`

```
static void Minisat::append (
    const vec< T > & from,
    vec< T > & to ) [inline], [static]
```

0.14.7.2.2 copy() [1/2] `template<class T >`

```
static void Minisat::copy (
    const T & from,
    T & to ) [inline], [static]
```

0.14.7.2.3 copy() [2/2] `template<class T >`

```
static void Minisat::copy (
    const vec< T > & from,
    vec< T > & to,
    bool append = false ) [inline], [static]
```

0.14.7.2.4 find() `template<class V , class T >`

```
static bool Minisat::find (
    V & ts,
    const T & t ) [inline], [static]
```

0.14.7.2.5 hash() [1/4] `static uint32_t Minisat::hash (`
`int32_t x) [inline], [static]`

0.14.7.2.6 hash() [2/4] `static uint32_t Minisat::hash (`
`int64_t x) [inline], [static]`

0.14.7.2.7 hash() [3/4] `static uint32_t Minisat::hash (`
`uint32_t x) [inline], [static]`

0.14.7.2.8 hash() [4/4] `static uint32_t Minisat::hash (`
`uint64_t x) [inline], [static]`

0.14.7.2.9 mkLit() `Lit` `Minisat::mkLit (`
`Var var,`
`bool sign = false) [inline]`

0.14.7.2.10 neg() `Lit` `Minisat::neg (`
`Lit p) [inline]`

0.14.7.2.11 operator<<() [1/5] std::ostream& Minisat::operator<< (std::ostream & os, const Lit & l) [inline]

0.14.7.2.12 operator<<() [2/5] std::ostream& Minisat::operator<< (std::ostream & os, const Minisat::Clause & c) [inline]

0.14.7.2.13 operator<<() [3/5] template<> std::ostream& Minisat::operator<< (std::ostream & os, const vec< char > & v) [inline]

0.14.7.2.14 operator<<() [4/5] template<typename T > std::ostream& Minisat::operator<< (std::ostream & os, const vec< T > & v) [inline]

0.14.7.2.15 operator<<() [5/5] std::ostream& Minisat::operator<< (std::ostream & os, Minisat::lbool b) [inline]

0.14.7.2.16 operator^() Lit Minisat::operator^ (Lit p, bool b) [inline]

0.14.7.2.17 operator~() Lit Minisat::operator~ (Lit p) [inline]

0.14.7.2.18 printUsageAndExit() void Minisat::printUsageAndExit (int argc, char ** argv, bool verbose = false)

0.14.7.2.19 remove() template<class V , class T > static void Minisat::remove (V & ts, const T & t) [inline], [static]

0.14.7.2.20 selectionSort() [1/2] template<class T > static void Minisat::selectionSort (T * array, int size) [inline], [static]

0.14.7.2.21 `selectionSort()` [2/2] template<class T , class LessThan >
void Minisat::selectionSort (

```
    T * array,
    int size,
    LessThan lt )
```

0.14.7.2.22 `setHelpPrefixStr()` void Minisat::setHelpPrefixStr (

```
    const char * str )
```

0.14.7.2.23 `setUsageHelp()` void Minisat::setUsageHelp (

```
    const char * str )
```

0.14.7.2.24 `sign()` bool Minisat::sign (

```
    Lit p ) [inline]
```

0.14.7.2.25 `sort()` [1/4] template<class T >
static void Minisat::sort (

```
    T * array,
    int size ) [inline], [static]
```

0.14.7.2.26 `sort()` [2/4] template<class T , class LessThan >
void Minisat::sort (

```
    T * array,
    int size,
    LessThan lt )
```

0.14.7.2.27 `sort()` [3/4] template<class T >
void Minisat::sort (

```
    vec< T > & v )
```

0.14.7.2.28 `sort()` [4/4] template<class T , class LessThan >
void Minisat::sort (

```
    vec< T > & v,
    LessThan lt )
```

0.14.7.2.29 `toInt()` [1/3] int Minisat::toInt (

```
    lbool l ) [inline]
```

0.14.7.2.30 `toInt()` [2/3] int Minisat::toInt (

```
    Lit p ) [inline]
```

0.14.7.2.31 `toInt()` [3/3] int Minisat::toInt (

```
    Var v ) [inline]
```

0.14.7.2.32 `toLbool()` `lbool` Minisat::toLbool (
 int v) [inline]

0.14.7.2.33 `toLit()` `Lit` Minisat::toLit (
 int i) [inline]

0.14.7.2.34 `var()` `int` Minisat::var (
 Lit p) [inline]

0.14.7.2.35 `xrealloc()` static void* Minisat::xrealloc (
 void * ptr,
 size_t size) [inline], [static]

0.14.7.3 Variable Documentation

0.14.7.3.1 `CONFLICT_CLAUSE` const unsigned Minisat::CONFLICT_CLAUSE = 2 [static]

0.14.7.3.2 `CRef_Lazy` const `CRef` Minisat::CRef_Lazy = `RegionAllocator<uint32_t>::Ref_Undef` - 1

0.14.7.3.3 `CRef_TPropagation` const `CRef` Minisat::CRef_TPropagation = `RegionAllocator<uint32_t>::Ref_Undef` - 2

0.14.7.3.4 `CRef_Undef` const `CRef` Minisat::CRef_Undef = `RegionAllocator<uint32_t>::Ref_Undef`

0.14.7.3.5 `LEMMA_CLAUSE` const unsigned Minisat::LEMMA_CLAUSE = 1 [static]

0.14.7.3.6 `lit_Error` const `Lit` Minisat::lit_Error = { -1 }

0.14.7.3.7 `lit_Undef` const `Lit` Minisat::lit_Undef = { -2 }

0.14.7.3.8 `NORMAL_CLAUSE` const unsigned Minisat::NORMAL_CLAUSE = 0 [static]

0.14.7.3.9 `nprimes` const int Minisat::nprimes = 25 [static]

0.14.7.3.10 `PERMANENT_CLAUSE` const unsigned Minisat::PERMANENT_CLAUSE = 3 [static]

0.14.7.3.11 `primes` const int Minisat::primes[nprimes] = { 31, 73, 151, 313, 643, 1291, 2593, 5233, 10501, 21013, 42073, 84181, 168451, 337219, 674701, 1349473, 2699299, 5398891, 10798093, 21596719, 43193641, 86387383, 172775299, 345550609, 691101253 } [static]

0.14.8 smtrat Namespace Reference

Implements a specialization of std::hash for any boost::variant.

Namespaces

- [analyzer](#)
- [cad](#)
- [cadcells](#)

A framework for sample-based CAD algorithms.

- [compile_information](#)

Compile time generated information about compiler and system version.

- [datastructures](#)
- [execution](#)
- [expression](#)
- [groebner](#)
- [helper](#)
- [icp](#)
- [impl](#)
- [lra](#)
- [lve](#)
- [maxsmt](#)

Contains strategy implementations for max SMT computations.

- [mcsat](#)
- [onecellcad](#)
- [options_detail](#)
- [parseformula](#)
- [parser](#)
- [preprocessor](#)
- [qe](#)
- [resource](#)
- [sat](#)
- [settings](#)
- [statistics](#)
- [subtropical](#)

Implements data structures and encodings for the subtropical method.

- [uf_impl](#)
- [unsatcore](#)

Contains strategy implementations for unsat core computations.

- [validation](#)
- [vb](#)
- [vs](#)

Data Structures

- struct [is_variant](#)

States whether a given type is a boost::variant.

- struct [is_variant< boost::variant< BOOST_VARIANT_ENUM_PARAMS\(U\) > >](#)

States that boost::variant is indeed a boost::variant.

- struct [is_variant< const boost::variant< BOOST_VARIANT_ENUM_PARAMS\(U\) > >](#)

States that const boost::variant is indeed a boost::variant.

- class [VariantMap](#)

This class is a specialization of std::map where the keys are of arbitrary type and the values are of type boost::variant.

- struct [CMakeOptionPrinter](#)

- class [Executor](#)
- class [PriorityQueue](#)
- class [DynamicPriorityQueue](#)
- class [SettingsComponents](#)
- class [SettingsParser](#)
- class [MaxSMT](#)
- class [BEModule](#)
- class [BVDirectEncoder](#)
- class [BVMODULE](#)
- class [CNFerModule](#)
- class [CoCoAGBModule](#)
- class [Bimap](#)

Container that stores expensive to construct objects and allows the fast lookup with respect to two independent keys within the objects.

- class [CSplitModule](#)
- class [CubeLIAModule](#)
- class [CurryModule](#)
- class [EMMModule](#)
- class [freelist](#)

Implements a fixed-(at-runtime)-size allocator based on a free list embedded in the space of currently unused objects.

- class [fixedsize_freelist](#)

Implements a fixed-(at-compile-time)-size allocator based on a free list embedded in the space of currently unused objects.

- class [fixedsize_allocator](#)

An allocator meant to be used for containers that typically allocate single objects or nodes, such as sets, maps and linked lists.

- class [fixedsize_allocator< void >](#)

I do not know where an allocator for void would be of any use; but in order to mimic std::allocator (where this is explicitly mentioned) as good as possible, we do this for the void case.

- class [dynarray](#)

dynarray is similar to std::vector, but has only a reduced interface.

- class [dynarray_allocator](#)
- class [union_find](#)

Implements a union-find datastructure, a datastructure containing an array of entries.

- class [EQModule](#)
- struct [by_address_hasher](#)

Hash an iterator by the pointed-to address.

- struct [pairhash](#)

Note that this may NOT be a specialization of std::hash for std::pair, this is not allowed by the standard as it does not involve a user-defined type.

- struct [variant_hash_visitor](#)
- struct [CollectBoolsInUEQs](#)
- struct [BoolUEQRewriter](#)
- class [EQPreprocessingModule](#)
- struct [UFRewriter](#)
- struct [JunctorMerger](#)
- class [ModuleWrapper](#)
- struct [NNFPreparation](#)
- class [NNFRewriter](#)
- struct [OrPathShortener](#)

A rewriter that checks every (X)OR that only contains ANDs and UEQs as children for equalities that hold for every alternative, and adds such equalities explicitly.

- struct [ReplaceVariablesRewriter](#)
- class [ESModule](#)

- class [FouMoModule](#)
A module which applies the Fourier-Motzkin algorithm.
- class [FPPModule](#)
- class [GBModule](#)
A solver module based on Groebner basis.
- class [GBModuleState](#)
A class to save the current state of a [GBModule](#).
- class [InequalitiesTable](#)
Datastructure for the [GBModule](#).
- class [VariableRewriteRule](#)
- class [GBPPModule](#)
- class [ForwardHyperGraph](#)
This class implements a forward hypergraph.
- struct [CycleEnumerator](#)
This class encapsulates an algorithm for enumerating all cycles.
- class [ICEModule](#)
- class [ICPModule](#)
- class [IncWidthModule](#)
- class [BVAnnotation](#)
- class [AnnotatedBVTerm](#)
- class [BlastedPoly](#)
- class [BlastedConstr](#)
- class [ConstrTree](#)
- class [ElementWithOrigins](#)
- class [CollectionWithOrigins](#)
- class [IntBlastModule](#)
- class [PolyTree](#)
- class [PolyTreeContent](#)
- class [PolyTreePool](#)
- class [IntEqModule](#)
A module which checks whether the equations contained in the received (linear integer) formula have a solution.
- class [LRAModule](#)
A module which performs the Simplex method on the linear part of it's received formula.
- class [LVEModule](#)
- class [MCBModule](#)
- class [NewCADModule](#)
- class [Backend](#)
- struct [SampledDerivationRefCompare](#)
- class [LevelWiseInformation](#)
- class [NewCoveringModule](#)
- struct [sampling](#)
- struct [is_sample_outside](#)
- struct [sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING >](#)
- struct [is_sample_outside< IsSampleOutsideAlgorithm::DEFAULT >](#)
- class [RationalCapsule](#)
- class [VariableCapsule](#)
- class [AxiomFactory](#)
- class [LOG](#)
- class [MonomialMappingByVariablePool](#)
- class [NRAILModule](#)
- class [PBGaussModule](#)
- class [CardinalityEncoder](#)
- class [ExactlyOneCommanderEncoder](#)

- class [LongFormulaEncoder](#)
- class [MixedSignEncoder](#)
- class [PseudoBoolEncoder](#)

Base class for a PseudoBoolean Encoder.
- class [PseudoBoolNormalizer](#)
- class [RNSEncoder](#)
- class [ShortFormulaEncoder](#)
- class [TotalizerEncoder](#)

TotalizerEncoder implements the Totalizer encoding as described in "Incremental Cardinality Constraints for MaxSAT" by Martins et al https://doi.org/10.1007/978-3-319-10428-7_39.
- class [TotalizerTree](#)
- class [PBPPModule](#)
- class [PFEModule](#)
- class [VarSchedulerMcsatBase](#)

Base class for all MCSAT variable scheduler.
- class [TheoryVarSchedulerStatic](#)

Schedules theory variables statically.
- class [VarSchedulerMcsatBooleanFirst](#)

Variable scheduling that all decides Boolean variables first before deciding any theory variable.
- class [VarSchedulerMcsatTheoryFirst](#)

Variable scheduling that all decides theory variables first before deciding any Boolean variable.
- class [VarSchedulerMcsatUnivariateConstraintsOnly](#)

Decides only Constraints univariate in the current theory variable while the theory ordering is static.
- class [VarSchedulerMcsatUnivariateClausesOnly](#)

Decides only Constraints occurring in clauses that are univariate in the current theory variable while the theory ordering is static.
- class [VarSchedulerMcsatActivityPreferTheory](#)

Activity-based scheduler preferring initially theory variables.
- class [SATModule](#)

Implements a module performing DPLL style SAT checking.
- class [VarSchedulerBase](#)

Base class for variable schedulers.
- class [VarSchedulerMinisat](#)

Minisat's activity-based variable scheduling.
- class [VarSchedulerFixedRandom](#)
- class [VarSchedulerRandom](#)

Random scheduler.
- class [VarSchedulerSMTTheoryGuided](#)

Scheduler for SMT, implementing theory guided heuristics.
- class [SplitSOSModule](#)
- class [STropModule](#)
- class [SymmetryModule](#)
- class [UFCegarModule](#)
- struct [EQGraph](#)
- struct [UnionFindInterface](#)
- struct [StaticUnionFind](#)
- struct [PersistentUnionFind](#)
- struct [Backtrackable](#)
- class [UnionFindModule](#)
- class [VSModule](#)
- class [Optimization](#)
- class [Manager](#)

- **Base class for solvers.**
- struct **Branching**
Stores all necessary information of an branch, which can be used to detect probable infinite loop of branchings.
- class **Module**
A base class for all kind of theory solving methods.
- class **FormulaWithOrigins**
Stores a formula along with its origins.
- class **ModuleInput**
The input formula a module has to consider for it's satisfiability check.
- class **PModule**
- struct **AbstractModuleFactory**
- struct **ModuleFactory**
- class **BackendLink**
- class **StrategyGraph**
- class **Task**
- class **BackendSynchronisation**
- class **ThreadPool**
- class **UnsatCore**

Typedefs

- using **ModelVariable** = carl::ModelVariable
- using **ModelValue** = carl::ModelValue< Rational, Poly >
- using **Model** = carl::Model< Rational, Poly >
- using **ModelSubstitution** = carl::ModelSubstitution< Rational, Poly >
- using **ModelPolynomialSubstitution** = carl::ModelPolynomialSubstitution< Rational, Poly >
- using **InfinityValue** = carl::InfinityValue
- using **SqrtEx** = carl::SqrtEx< Poly >
- using **MultivariateRootT** = carl::MultivariateRoot< Poly >
- using **DoubleInterval** = carl::Interval< double >
- using **RationalInterval** = carl::Interval< Rational >
- using **EvalDoubleIntervalMap** = std::map< carl::Variable, DoubleInterval >
- using **EvalRationalIntervalMap** = std::map< carl::Variable, RationalInterval >
- using **ObjectiveValues** = std::vector< std::pair< std::variant< Poly, std::string >, Model::mapped_type > >
- using **Statistics** = carl::statistics::Statistics
- using **Conditionals** = std::vector< std::atomic_bool * >
A vector of atomic bool pointers.
- using **Rational** = mpq_class
- using **Integer** = carl::IntegralType< Rational >::type
- using **TermT** = carl::Term< Rational >
- using **Poly** = carl::MultivariatePolynomial< Rational >
- using **Factorization** = std::map< Poly, carl::uint >
- using **ConstraintT** = carl::Constraint< Poly >
- using **ConstraintsT** = carl::Constraints< Poly >
- using **VariableAssignmentT** = carl::VariableAssignment< Poly >
- using **VariableComparisonT** = carl::VariableComparison< Poly >
- using **FormulaT** = carl::Formula< Poly >
- using **FormulasT** = carl::Formulas< Poly >
- using **FormulaSetT** = carl::FormulaSet< Poly >
- using **FormulasMultiT** = std::multiset< FormulaT >
- using **RationalAssignment** = carl::Assignment< Rational >
- using **RAN** = carl::IntRepRealAlgebraicNumber< Rational >
- using **thread_priority** = std::pair< std::size_t, std::size_t >
- typedef expression::Expression Expression

- `typedef expression::Expressions Expressions`
- `typedef std::pair< FormulaT, std::shared_ptr< std::vector< FormulaT > >> SingleFormulaOrigins`
- `typedef std::map< FormulaT, std::shared_ptr< std::vector< FormulaT > >> FormulaOrigins`
- `typedef std::map< carl::Variable, std::pair< std::vector< SingleFormulaOrigins >, std::vector< SingleFormulaOrigins > >> VariableUpperLower`
- `typedef std::set< icp::ContractionCandidate *, icp::contractionCandidateComp > ContractionCandidates`
- `typedef std::map< FormulaT, std::shared_ptr< std::vector< FormulaT > >> Formula_Origins`
- `typedef std::unordered_map< carl::Variable, carl::Monomial::Arg >::iterator MonomialMapIterator`
- `typedef std::unordered_map< carl::Variable, carl::Monomial::Arg > MonomialMap`
- `using Type = carl::UVariable`
- `template<typename Impl>`
`using EQClasses = UnionFindInterface< Type, Impl >`
- `using StaticClasses = EQClasses< StaticUnionFind >`
- `using BacktrackableClasses = EQClasses< Backtrackable< PersistentUnionFind > >`
- `using Classes = BacktrackableClasses`
- `typedef bool(* ConditionEvaluation) (carl::Condition)`
- `typedef std::function< bool(carl::Condition)> ConditionFunction`
- `using Priority = std::vector< std::size_t >`

Enumerations

- `enum LemmaLevel { NONE = 0 , NORMAL = 1 , ADVANCED = 2 }`
- `enum Answer { SAT = 0 , UNSAT = 1 , UNKNOWN = 2 , ABORTED = 3 , OPTIMAL = 4 }`

An enum with the possible answers a Module can give.
- `enum class MaxSMTStrategy { FU_MALIK_INCREMENTAL , MSU3 , LINEAR_SEARCH }`
- `enum CoveringStatus { partial = 0 , full = 1 , unknown = 2 , failed = 3 }`
- `enum SamplingAlgorithm { LOWER_UPPER_BETWEEN_SAMPLING }`
- `enum IsSampleOutsideAlgorithm { DEFAULT }`
- `enum class TheoryGuidedDecisionHeuristicLevel : unsigned { CONFLICT_FIRST , SATISFIED_FIRST , DISABLED }`
- `enum class UnsatCoreStrategy { ModelExclusion }`

Functions

- `int handle_basic_options (const SettingsParser &parser)`
- `const auto & settings_parser ()`
- `bool parseSMT2File (parser::InstructionHandler &handler, bool queueInstructions, std::istream &input)`
- `int convert_to_cnf_dimacs (const std::string &, const std::string &)`
- `int convert_to_cnf_smtlib (const std::string &, const std::string &)`
- `std::ostream & operator<< (std::ostream &os, CMakeOptionPrinter cmop)`
- `constexpr CMakeOptionPrinter CMakeOptions (bool advanced=false) noexcept`
- `template<typename Executor>`
`bool parseInput (const std::string &pathToInputFile, Executor &e, bool &queueInstructions)`
- `template<typename Executor>`
`int executeFile (const std::string &pathToInputFile, Executor &e)`

Parse the file and save it in formula.
- `int analyze_file (const std::string &)`
- `template<typename Strategy>`
`int run_dimacs_file (Strategy &, const std::string &)`
- `template<typename Strategy>`
`int run_opb_file (Strategy &, const std::string &)`
- `FormulaT parse_smtlib (const std::string &)`
- `FormulaT parse_formula (const std::string &filename)`

- Loads the smt2 file specified in filename and returns the formula.*
- int `preprocess_file` (const std::string &filename, const std::string &outfile)

Loads the smt2 file specified in filename and runs the preprocessing strategy.
 - const auto & `settings_analyzer` ()
 - `analyzer::AnalyzerStatistics & analyze_formula` (const `FormulaT` &f)
 - template<typename TT, typename C>
 std::ostream & `operator<<` (std::ostream &os, const `PriorityQueue< TT, C >` &pq)
 - const `settings::Settings & Settings` ()
 - const auto & `settings_core` ()
 - const auto & `settings_solver` ()
 - const auto & `settings_module` ()
 - template<typename T>
 auto & `statistics_get` (const std::string &name)
 - const auto & `settings_statistics` ()
 - bool `is_sat` (Answer a)
 - std::ostream & `operator<<` (std::ostream &os, Answer a)
 - auto & `validation_get` (const std::string &channel, const std::string &file, int line)
 - const auto & `settings_validation` ()
 - void `visiting` (const `Expression` &expr)
 - void `testExpression` ()
 - std::ostream & `operator<<` (std::ostream &os, `MaxSMTStrategy` ucs)
 - static void `hash_combine` (std::size_t &seed, std::size_t hash_value) noexcept
 - std::ostream & `operator<<` (std::ostream &os, const `CoveringStatus` &status)
 - template<typename Settings>
 std::ostream & `operator<<` (std::ostream &os, const `LevelWiseInformation< Settings >` &levelWiseInformation)
 - std::string `get_name` (`SamplingAlgorithm` samplingAlgorithm)
 - std::string `get_name` (`IsSampleOutsideAlgorithm` samplingAlgorithm)
 - `FormulaT createZeroOne` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable)
 - `FormulaT createZeroTwo` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable)
 - `FormulaT createZeroThree` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable)
 - `FormulasT createZero` (`smrat::VariableCapsule` variableCapsule)
 - `FormulaT createTangentPlaneNEQOne` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable, Rational aRational)
 - `FormulaT createTangentPlaneNEQTTwo` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable, Rational bRational)
 - `FormulaT createTangentPlaneNEQThree` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable, Rational aRational, Rational bRational)
 - `FormulaT createTangentPlaneNEQFour` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable, Rational aRational, Rational bRational)
 - `FormulasT createTangentPlaneNEQ` (`VariableCapsule` variableCapsule, `RationalCapsule` rationalCapsule)
 - `FormulaT createTangentPlaneEQOne` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable, Rational aRational)
 - `FormulaT createTangentPlaneEQTTwo` (carl::Variable xVariable, carl::Variable yVariable, carl::Variable zVariable, Rational aRational)
 - `FormulasT createTangentPlaneEQ` (`VariableCapsule` variableCapsule, `RationalCapsule` rationalCapsule)
 - const `smrat::VariableCapsule extractVariables` (`MonomialMapIterator` monomialIterator)
 - `Model createAbsoluteValuedModel` (`Model` linearizedModel)
 - carl::Variable `createAuxiliaryVariable` (carl::Variable variable)

create an auxiliary variable e.g.

 - `FormulaT generateAbsFormula` (carl::Variable variable, carl::Variable aux_variable)

| x1 | = (and (=> (x1 >= 0) (y1 = x1)) (=> (x1 < 0) (y1 = -x1)))
 - `FormulaT generateAbsFormula` (carl::Variable variableLeft, carl::Variable variableRight, carl::Relation relation)

-
- ```

| x1 | <= | x1 | = (and (=> (x1 >= 0) (y1 = x1)) (=> (x1 < 0) (y1 = -x1)) (=> (x2 >= 0) (y2 = x2)) (=> (x2 < 0) (y2 = -x2)) (y1 <= y2))
• FormulaT createEquivalentToOriginalMonotonicityOne (smrat::VariableCapsule variableCapsule, smrat::VariableCapsule variableCapsuleInnner)
• FormulaT createEquivalentToOriginalMonotonicityTwo (smrat::VariableCapsule variableCapsule, smrat::VariableCapsule variableCapsuleInnner)
• FormulaT createEquivalentToOriginalMonotonicityThree (smrat::VariableCapsule variableCapsule, smrat::VariableCapsule variableCapsuleInnner)
• RationalCapsule extractRationalCapsule (VariableCapsule variableCapsule, Model linearizedModel)
• FormulaT createOriginalMonotonicityOne (smrat::VariableCapsule variableCapsule, smrat::VariableCapsule variableCapsuleInnner)
• FormulaT createOriginalMonotonicityTwo (smrat::VariableCapsule variableCapsule, smrat::VariableCapsule variableCapsuleInnner)
• FormulaT createOriginalMonotonicityThree (smrat::VariableCapsule variableCapsule, smrat::VariableCapsule variableCapsuleInnner)
• FormulasT createMonotonicity (VariableCapsule variableCapsuleOuter, VariableCapsule variableCapsuleInner, Model absoluteValuedModel)
• FormulaT createCongruence (smrat::VariableCapsule variableCapsuleOuter, smrat::VariableCapsule variableCapsuleInnner)
• FormulaT createICPGreaterOne (VariableCapsule variableCapsule, RationalCapsule rationalCapsule)
• FormulaT createICPGreaterTwo (VariableCapsule variableCapsule, RationalCapsule rationalCapsule)
• FormulasT createICPGreater (VariableCapsule variableCapsule, RationalCapsule rationalCapsule)
• FormulaT createICPLess (VariableCapsule variableCapsule, RationalCapsule rationalCapsule)
• bool abEqualcCheck (VariableCapsule variableCapsuleOuter, Model linearizedModel)
• bool abGreatercCheck (RationalCapsule rationalCapsule)
• bool abLesscCheck (RationalCapsule rationalCapsule)
• RationalCapsule generateAbcPrimeForICP (RationalCapsule rationalCapsule, bool isGreater)
• bool isAnyRationalIsZero (RationalCapsule rationalCapsule)
• Rational factorial (Rational n)
• Rational factorial (std::size_t)
• std::ostream & operator<< (std::ostream &os, Module::LemmaType lt)
• template<typename AnnotationType>
void annotateFormula (const FormulaT &, const std::vector<AnnotationType> &)
• std::ostream & operator<< (std::ostream &os, const ModuleInput &mi)
• bool isCondition (carl::Condition _condition)
• template<typename Module>
std::ostream & operator<< (std::ostream &os, const ModuleFactory<Module> &mf)
• std::ostream & operator<< (std::ostream &os, UnsatCoreStrategy ucs)

```

## Variables

- static const Model EMPTY\_MODEL = Model()
- static constexpr int COMMANDER\_GROUP\_SIZE = 4

### 0.14.8.1 Detailed Description

Implements a specialization of std::hash for any boost::variant.

Class to create a substitution object.

Class to create a state object.

Class to create a condition object.

Class to create the formulas for axioms.

This only works if the types of the variant are std::hash-enabled (have a std::hash specialization of their own).

## Author

Aklima Zaman

Since

2018-09-24

Version

2018-09-24

Author

Florian Corzilius

Since

2010-07-26

Version

2011-06-20

Author

Florian Corzilius

Since

2010-07-26

Version

2011-12-05

Author

Florian Corzilius

Since

2010-05-11

Version

2013-10-24

Author

Florian Corzilius

Since

2010-05-11

Version

2013-10-23

## 0.14.8.2 Typedef Documentation

**0.14.8.2.1 BacktrackableClasses** `using smtrat::BacktrackableClasses = typedef EQClasses< Backtrackable< PersistentUnionFind > >`

**0.14.8.2.2 Classes** using `smtrat::Classes` = `typedef BacktrackableClasses`

**0.14.8.2.3 Conditionals** `typedef std::vector< std::atomic_bool * > smtrat::Conditionals`  
A vector of atomic bool pointers.

**0.14.8.2.4 ConditionEvaluation** `typedef bool(* smtrat::ConditionEvaluation) (carl::Condition)`

**0.14.8.2.5 ConditionFunction** `typedef std::function<bool(carl::Condition)> smtrat::ConditionFunction`

**0.14.8.2.6 ConstraintsT** using `smtrat::ConstraintsT` = `typedef carl::Constraints<Poly>`

**0.14.8.2.7 ConstraintT** using `smtrat::ConstraintT` = `typedef carl::Constraint<Poly>`

**0.14.8.2.8 ContractionCandidates** `typedef std::set<icp::ContractionCandidate*, icp::contractionCandidateComp> smtrat::ContractionCandidates`

**0.14.8.2.9 DoubleInterval** using `smtrat::DoubleInterval` = `typedef carl::Interval<double>`

**0.14.8.2.10 EQClasses** template<typename Impl >  
using `smtrat::EQClasses` = `typedef UnionFindInterface< Type, Impl >`

**0.14.8.2.11 EvalDoubleIntervalMap** using `smtrat::EvalDoubleIntervalMap` = `typedef std::map<carl::Variable, DoubleInterval>`

**0.14.8.2.12 EvalRationalIntervalMap** using `smtrat::EvalRationalIntervalMap` = `typedef std::map<carl::Variable, RationalInterval>`

**0.14.8.2.13 Expression** `typedef expression::Expression smtrat::Expression`

**0.14.8.2.14 Expressions** `typedef expression::Expressions smtrat::Expressions`

**0.14.8.2.15 Factorization** using `smtrat::Factorization` = `typedef std::map<Poly, carl::uint>`

**0.14.8.2.16 Formula\_Origins** `typedef std::map< FormulaT, std::shared_ptr< std::vector<FormulaT> > > smtrat::Formula_Origins`

**0.14.8.2.17 FormulaOrigins** `typedef std::map<FormulaT, std::shared_ptr<std::vector<FormulaT> > > smtrat::FormulaOrigins`

**0.14.8.2.18 FormulaSetT** using `smtrat::FormulaSetT` = `typedef carl::FormulaSet<Poly>`

**0.14.8.2.19 FormulasMultiT** using `smtrat::FormulasMultiT` = `typedef std::multiset<FormulaT>`

**0.14.8.2.20 FormulasT** using `smtrat::FormulasT` = `typedef carl::Formulas<Poly>`

**0.14.8.2.21 FormulaT** using `smtrat::FormulaT` = `typedef carl::Formula<Poly>`

**0.14.8.2.22 InfinityValue** using `smtrat::InfinityValue` = `typedef carl::InfinityValue`

**0.14.8.2.23 Integer** using `smtrat::Integer` = `typedef carl::IntegralType<Rational>::type`

**0.14.8.2.24 Model** using `smtrat::Model` = `typedef carl::Model<Rational, Poly>`

**0.14.8.2.25 ModelPolynomialSubstitution** using `smtrat::ModelPolynomialSubstitution` = `typedef carl::ModelPolynomialSubstitution<Rational, Poly>`

**0.14.8.2.26 ModelSubstitution** using `smtrat::ModelSubstitution` = `typedef carl::ModelSubstitution<Rational, Poly>`

**0.14.8.2.27 ModelValue** using `smtrat::ModelValue` = `typedef carl::ModelValue<Rational, Poly>`

**0.14.8.2.28 ModelVariable** using `smtrat::ModelVariable` = `typedef carl::ModelVariable`

**0.14.8.2.29 MonomialMap** `typedef std::unordered_map<carl::Variable, carl::Monomial::Arg>`  
`smtrat::MonomialMap`

**0.14.8.2.30 MonomialMapIterator** `typedef std::unordered_map<carl::Variable, carl::Monomial::Arg>::iterator`  
`smtrat::MonomialMapIterator`

**0.14.8.2.31 MultivariateRootT** using `smtrat::MultivariateRootT` = `typedef carl::MultivariateRoot<Poly>`

**0.14.8.2.32 ObjectiveValues** using `smtrat::ObjectiveValues` = `typedef std::vector<std::pair<std::variant<Poly, std::string>, Model::mapped_type>>`

**0.14.8.2.33 Poly** using `smtrat::Poly` = `typedef carl::MultivariatePolynomial<Rational>`

**0.14.8.2.34 Priority** using `smtrat::Priority` = `typedef std::vector<std::size_t>`

**0.14.8.2.35 RAN** using `smtrat::RAN` = `typedef carl::IntRepRealAlgebraicNumber<Rational>`

**0.14.8.2.36 Rational** using `smtrat::Rational` = `typedef mpq_class`

**0.14.8.2.37 RationalAssignment** using `smtrat::RationalAssignment` = `typedef carl::Assignment<Rational>`

**0.14.8.2.38 RationalInterval** using `smtrat::RationalInterval` = `typedef carl::Interval<Rational>`

**0.14.8.2.39 SingleFormulaOrigins** `typedef std::pair<FormulaT, std::shared_ptr<std::vector<FormulaT>> > > smtrat::SingleFormulaOrigins`

**0.14.8.2.40 SqrtEx** using `smtrat::SqrtEx` = `typedef carl::SqrtEx<Poly>`

**0.14.8.2.41 StaticClasses** using `smtrat::StaticClasses` = `typedef EQClasses< StaticUnionFind >`

**0.14.8.2.42 Statistics** using `smtrat::Statistics` = `typedef carl::statistics::Statistics`

**0.14.8.2.43 TermT** using `smtrat::TermT` = `typedef carl::Term<Rational>`

**0.14.8.2.44 thread\_priority** using `smtrat::thread_priority` = `typedef std::pair<std::size_t, std::size_t>`

**0.14.8.2.45 Type** using `smtrat::Type` = `typedef carl::UVariable`

**0.14.8.2.46 VariableAssignmentT** using `smtrat::VariableAssignmentT` = `typedef carl::VariableAssignment<Poly>`

**0.14.8.2.47 VariableComparisonT** using `smtrat::VariableComparisonT` = `typedef carl::VariableComparison<Poly>`

**0.14.8.2.48 VariableUpperLower** `typedef std::map<carl::Variable, std::pair< std::vector<SingleFormulaOrigins>, std::vector<SingleFormulaOrigins>> > > smtrat::VariableUpperLower`

## 0.14.8.3 Enumeration Type Documentation

**0.14.8.3.1 Answer** `enum smtrat::Answer`

An enum with the possible answers a [Module](#) can give.

Enumerator

|         |
|---------|
| SAT     |
| UNSAT   |
| UNKNOWN |
| ABORTED |
| OPTIMAL |

#### 0.14.8.3.2 CoveringStatus `enum smtrat::CoveringStatus`

Enumerator

|         |
|---------|
| partial |
| full    |
| unknown |
| failed  |

#### 0.14.8.3.3 IsSampleOutsideAlgorithm `enum smtrat::IsSampleOutsideAlgorithm`

Enumerator

|         |
|---------|
| DEFAULT |
|---------|

#### 0.14.8.3.4 LemmaLevel `enum smtrat::LemmaLevel`

Enumerator

|          |
|----------|
| NONE     |
| NORMAL   |
| ADVANCED |

#### 0.14.8.3.5 MaxSMTStrategy `enum smtrat::MaxSMTStrategy [strong]`

Enumerator

|                      |
|----------------------|
| FU_MALIK_INCREMENTAL |
| MSU3                 |
| LINEAR_SEARCH        |

#### 0.14.8.3.6 SamplingAlgorithm `enum smtrat::SamplingAlgorithm`

Enumerator

|                              |
|------------------------------|
| LOWER_UPPER_BETWEEN_SAMPLING |
|------------------------------|

**0.14.8.3.7 TheoryGuidedDecisionHeuristicLevel** enum `smtrat::TheoryGuidedDecisionHeuristicLevel`  
: unsigned [strong]

Enumerator

|                 |  |
|-----------------|--|
| CONFFLICT_FIRST |  |
| SATISFIED_FIRST |  |
| DISABLED        |  |

**0.14.8.3.8 UnsatCoreStrategy** enum `smtrat::UnsatCoreStrategy` [strong]

Enumerator

|                |  |
|----------------|--|
| ModelExclusion |  |
|----------------|--|

#### 0.14.8.4 Function Documentation

**0.14.8.4.1 abEqualcCheck()** bool `smtrat::abEqualcCheck` (  
    *VariableCapsule* *variableCapsuleOuter*,  
    *Model* *linearizedModel* )

**0.14.8.4.2 abGreatercCheck()** bool `smtrat::abGreatercCheck` (  
    *RationalCapsule* *rationalCapsule* )

**0.14.8.4.3 abLesscCheck()** bool `smtrat::abLesscCheck` (  
    *RationalCapsule* *rationalCapsule* )

**0.14.8.4.4 analyze\_file()** int `smtrat::analyze_file` (  
    const `std::string` & )

**0.14.8.4.5 analyze\_formula()** `analyzer::AnalyzerStatistics` & `smtrat::analyze_formula` (  
    const `FormulaT` & *f* )

**0.14.8.4.6 annotateFormula()** template<typename AnnotationType >  
void `smtrat::annotateFormula` (  
    const `FormulaT` & ,  
    const `std::vector< AnnotationType >` & )

**0.14.8.4.7 CMakeOptions()** constexpr `CMakeOptionPrinter` `smtrat::CMakeOptions` (  
    bool *advanced* = *false* ) [constexpr], [noexcept]

**0.14.8.4.8 convert\_to\_cnf\_dimacs()** `int smtrat::convert_to_cnf_dimacs (`  
`const std::string & ,`  
`const std::string & )`

**0.14.8.4.9 convert\_to\_cnf\_smtlib()** `int smtrat::convert_to_cnf_smtlib (`  
`const std::string & ,`  
`const std::string & )`

**0.14.8.4.10 createAbsoluteValuedModel()** `Model smtrat::createAbsoluteValuedModel (`  
`Model linearizedModel )`

**0.14.8.4.11 createAuxiliaryVariable()** `carl::Variable smtrat::createAuxiliaryVariable (`

`carl::Variable variable )`

create an auxiliary variable e.g.

`aux_<Variable Name>`

#### Parameters

|                       |                                                         |
|-----------------------|---------------------------------------------------------|
| <code>variable</code> | the variable for which auxiliary variable to be created |
|-----------------------|---------------------------------------------------------|

#### Returns

created auxiliary variable

**0.14.8.4.12 createCongruence()** `FormulaT smtrat::createCongruence (`  
`smtrat::VariableCapsule variableCapsuleOuter,`  
`smtrat::VariableCapsule variableCapsuleInner )`

**0.14.8.4.13 createEquivalentToOriginalMonotonicityOne()** `FormulaT smtrat::createEquivalentTo←`  
`OriginalMonotonicityOne (`  
`smtrat::VariableCapsule variableCapsule,`  
`smtrat::VariableCapsule variableCapsuleInner )`

**0.14.8.4.14 createEquivalentToOriginalMonotonicityThree()** `FormulaT smtrat::createEquivalentTo←`  
`OriginalMonotonicityThree (`  
`smtrat::VariableCapsule variableCapsule,`  
`smtrat::VariableCapsule variableCapsuleInner )`

**0.14.8.4.15 createEquivalentToOriginalMonotonicityTwo()** `FormulaT smtrat::createEquivalentTo←`  
`OriginalMonotonicityTwo (`  
`smtrat::VariableCapsule variableCapsule,`  
`smtrat::VariableCapsule variableCapsuleInner )`

**0.14.8.4.16 createICPGreater()** `Formulast smtrat::createICPGreater (`  
`VariableCapsule variableCapsule,`  
`RationalCapsule rationalCapsule )`

**0.14.8.4.17 `createICPGreaterOne()`** `FormulaT smtrat::createICPGreaterOne (`  
`VariableCapsule variableCapsule,`  
`RationalCapsule rationalCapsule )`

**0.14.8.4.18 `createICPGreaterTwo()`** `FormulaT smtrat::createICPGreaterTwo (`  
`VariableCapsule variableCapsule,`  
`RationalCapsule rationalCapsule )`

**0.14.8.4.19 `createICPLess()`** `FormulaT smtrat::createICPLess (`  
`VariableCapsule variableCapsule,`  
`RationalCapsule rationalCapsule )`

**0.14.8.4.20 `createMonotonicity()`** `FormulasT smtrat::createMonotonicity (`  
`VariableCapsule variableCapsuleOuter,`  
`VariableCapsule variableCapsuleInner,`  
`Model absoluteValuedModel )`

**0.14.8.4.21 `createOriginalMonotonicityOne()`** `FormulaT smtrat::createOriginalMonotonicityOne (`  
`smtrat::VariableCapsule variableCapsule,`  
`smtrat::VariableCapsule variableCapsuleInner )`

**0.14.8.4.22 `createOriginalMonotonicityThree()`** `FormulaT smtrat::createOriginalMonotonicityThree (`  
`smtrat::VariableCapsule variableCapsule,`  
`smtrat::VariableCapsule variableCapsuleInner )`

**0.14.8.4.23 `createOriginalMonotonicityTwo()`** `FormulaT smtrat::createOriginalMonotonicityTwo (`  
`smtrat::VariableCapsule variableCapsule,`  
`smtrat::VariableCapsule variableCapsuleInner )`

**0.14.8.4.24 `createTangentPlaneEQ()`** `FormulasT smtrat::createTangentPlaneEQ (`  
`VariableCapsule variableCapsule,`  
`RationalCapsule rationalCapsule )`

**0.14.8.4.25 `createTangentPlaneEQOne()`** `FormulaT smtrat::createTangentPlaneEQOne (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable,`  
`Rational aRational )`

**0.14.8.4.26 `createTangentPlaneEQTwo()`** `FormulaT smtrat::createTangentPlaneEQTwo (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable,`  
`Rational aRational )`

**0.14.8.4.27 `createTangentPlaneNEQ()`** `FormulasT smtrat::createTangentPlaneNEQ (`  
`VariableCapsule variableCapsule,`  
`RationalCapsule rationalCapsule )`

**0.14.8.4.28 `createTangentPlaneNEQFour()`** `FormulaT smtrat::createTangentPlaneNEQFour (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable,`  
`Rational aRational,`  
`Rational bRational )`

**0.14.8.4.29 `createTangentPlaneNEQOne()`** `FormulaT smtrat::createTangentPlaneNEQOne (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable,`  
`Rational aRational )`

**0.14.8.4.30 `createTangentPlaneNEQThree()`** `FormulaT smtrat::createTangentPlaneNEQThree (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable,`  
`Rational aRational,`  
`Rational bRational )`

**0.14.8.4.31 `createTangentPlaneNEQTwo()`** `FormulaT smtrat::createTangentPlaneNEQTwo (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable,`  
`Rational bRational )`

**0.14.8.4.32 `createZero()`** `FormulasT smtrat::createZero (`  
`smtrat::VariableCapsule variableCapsule )`

**0.14.8.4.33 `createZeroOne()`** `FormulaT smtrat::createZeroOne (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable )`

**0.14.8.4.34 `createZeroThree()`** `FormulaT smtrat::createZeroThree (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable )`

**0.14.8.4.35 `createZeroTwo()`** `FormulaT smtrat::createZeroTwo (`  
`carl::Variable xVariable,`  
`carl::Variable yVariable,`  
`carl::Variable zVariable )`

```
0.14.8.4.36 executeFile() template<typename Executor>
int smtrat::executeFile (
 const std::string & pathToInputFile,
 Executor & e)
```

Parse the file and save it in formula.

#### Parameters

|                        |                                                                          |
|------------------------|--------------------------------------------------------------------------|
| <i>pathToInputFile</i> | The path to the input smt2 file.                                         |
| <i>formula</i>         | A pointer to the formula object which holds the parsed input afterwards. |
| <i>options</i>         | Save options from the smt2 file here.                                    |

```
0.14.8.4.37 extractRationalCapsule() RationalCapsule smtrat::extractRationalCapsule (
 VariableCapsule variableCapsule,
 Model linearizedModel)
```

```
0.14.8.4.38 extractVariables() const smtrat::VariableCapsule smtrat::extractVariables (
 MonomialMapIterator monomialIterator)
```

```
0.14.8.4.39 factorial() [1/2] Rational smtrat::factorial (
 Rational n)
```

```
0.14.8.4.40 factorial() [2/2] Rational smtrat::factorial (
 std::size_t n)
```

```
0.14.8.4.41 generateAbcPrimeForICP() RationalCapsule smtrat::generateAbcPrimeForICP (
 RationalCapsule rationalCapsule,
 bool isGreater)
```

```
0.14.8.4.42 generateAbsFormula() [1/2] FormulaT smtrat::generateAbsFormula (
 carl::Variable variable,
 carl::Variable aux_variable)
| x1 | = (and (=> (x1 >= 0) (y1 = x1)) (=> (x1 < 0) (y1 = -x1)))
```

#### Parameters

|                 |  |
|-----------------|--|
| <i>variable</i> |  |
|-----------------|--|

#### Returns

```
0.14.8.4.43 generateAbsFormula() [2/2] FormulaT smtrat::generateAbsFormula (
 carl::Variable variableLeft,
 carl::Variable variableRight,
 carl::Relation relation)
| x1 | <= | x1 | = (and (=> (x1 >= 0) (y1 = x1)) (=> (x1 < 0) (y1 = -x1)) (=> (x2 >= 0) (y2 = x2)) (=> (x2 < 0) (y2 = -x2)) (y1 <= y2))
```

**Parameters**

|                            |  |
|----------------------------|--|
| <code>variableLeft</code>  |  |
| <code>variableRight</code> |  |
| <code>relation</code>      |  |

**Returns**

**0.14.8.4.44 `get_name()` [1/2]** `std::string smtrat::get_name (`  
`IsSampleOutsideAlgorithm samplingAlgorithm )` [inline]

**0.14.8.4.45 `get_name()` [2/2]** `std::string smtrat::get_name (`  
`SamplingAlgorithm samplingAlgorithm )` [inline]

**0.14.8.4.46 `handle_basic_options()`** `int smtrat::handle_basic_options (`  
`const SettingsParser & parser )`

**0.14.8.4.47 `hash_combine()`** `static void smtrat::hash_combine (`  
`std::size_t & seed,`  
`std::size_t hash_value )` [inline], [static], [noexcept]

**0.14.8.4.48 `is_sat()`** `bool smtrat::is_sat (`  
`Answer a )` [inline]

**0.14.8.4.49 `isAnyRationalIsZero()`** `bool smtrat::isAnyRationalIsZero (`  
`RationalCapsule rationalCapsule )`

**0.14.8.4.50 `isCondition()`** `bool smtrat::isCondition (`  
`carl::Condition _condition )`

**0.14.8.4.51 `operator<<()` [1/10]** `std::ostream& smtrat::operator<< (`  
`std::ostream & os,`  
`Answer a )` [inline]

**0.14.8.4.52 `operator<<()` [2/10]** `std::ostream & smtrat::operator<< (`  
`std::ostream & os,`  
`CMakeOptionPrinter cmop )`

**0.14.8.4.53 `operator<<()` [3/10]** `std::ostream& smtrat::operator<< (`  
`std::ostream & os,`  
`const CoveringStatus & status )` [inline]

**0.14.8.4.54 operator<<()** [4/10] template<typename Settings >  
std::ostream& smtrat::operator<< (  
 std::ostream & os,  
 const LevelWiseInformation< Settings > & levelWiseInformation ) [inline]

**0.14.8.4.55 operator<<()** [5/10] template<typename Module >  
std::ostream& smtrat::operator<< (  
 std::ostream & os,  
 const ModuleFactory< Module > & mf ) [inline]

**0.14.8.4.56 operator<<()** [6/10] std::ostream& smtrat::operator<< (  
 std::ostream & os,  
 const ModuleInput & mi ) [inline]

**0.14.8.4.57 operator<<()** [7/10] template<typename TT , typename C >  
std::ostream& smtrat::operator<< (  
 std::ostream & os,  
 const PriorityQueue< TT, C > & pq )

**0.14.8.4.58 operator<<()** [8/10] std::ostream& smtrat::operator<< (  
 std::ostream & os,  
 MaxSMTStrategy ucs ) [inline]

**0.14.8.4.59 operator<<()** [9/10] std::ostream& smtrat::operator<< (  
 std::ostream & os,  
 Module::LemmaType lt ) [inline]

**0.14.8.4.60 operator<<()** [10/10] std::ostream& smtrat::operator<< (  
 std::ostream & os,  
 UnsatCoreStrategy ucs ) [inline]

**0.14.8.4.61 parse\_formula()** FormulaT smtrat::parse\_formula (  
 const std::string & filename )

Loads the smt2 file specified in filename and returns the formula.

This function ignores most SMT-LIB commands but simply accumulated all asserted formulas.

**0.14.8.4.62 parse\_smtlib()** FormulaT smtrat::parse\_smtlib (  
 const std::string & )

**0.14.8.4.63 parseInput()** template<typename Executor >  
bool smtrat::parseInput (  
 const std::string & pathToInputFile,  
 Executor & e,  
 bool & queueInstructions )

```
0.14.8.4.64 parseSMT2File() bool smtrat::parseSMT2File (
 parser::InstructionHandler & handler,
 bool queueInstructions,
 std::istream & input)
```

```
0.14.8.4.65 preprocess_file() int smtrat::preprocess_file (
 const std::string & filename,
 const std::string & outfile)
```

Loads the smt2 file specified in filename and runs the preprocessing strategy.  
The resulting (simplified) problem is written to outfile (or stdout if outfile is empty).

```
0.14.8.4.66 run_dimacs_file() template<typename Strategy >
int smtrat::run_dimacs_file (
 Strategy & ,
 const std::string &)
```

```
0.14.8.4.67 run_opb_file() template<typename Strategy >
int smtrat::run_opb_file (
 Strategy & ,
 const std::string &)
```

```
0.14.8.4.68 Settings() const settings::Settings& smtrat::Settings () [inline]
```

```
0.14.8.4.69 settings_analyzer() const auto& smtrat::settings_analyzer () [inline]
```

```
0.14.8.4.70 settings_core() const auto& smtrat::settings_core () [inline]
```

```
0.14.8.4.71 settings_module() const auto& smtrat::settings_module () [inline]
```

```
0.14.8.4.72 settings_parser() const auto& smtrat::settings_parser () [inline]
```

```
0.14.8.4.73 settings_solver() const auto& smtrat::settings_solver () [inline]
```

```
0.14.8.4.74 settings_statistics() const auto& smtrat::settings_statistics () [inline]
```

```
0.14.8.4.75 settings_validation() const auto& smtrat::settings_validation () [inline]
```

```
0.14.8.4.76 statistics_get() template<typename T >
auto& smtrat::statistics_get (
 const std::string & name)
```

```
0.14.8.4.77 testExpression() void smtrat::testExpression ()
```

```
0.14.8.4.78 validation_get() auto& smrat::validation_get (
 const std::string & channel,
 const std::string & file,
 int line) [inline]
```

```
0.14.8.4.79 visiting() void smrat::visiting (
 const Expression & expr)
```

## 0.14.8.5 Variable Documentation

```
0.14.8.5.1 COMMANDER_GROUP_SIZE constexpr int smrat::COMMANDER_GROUP_SIZE = 4 [static],
[constexpr]
```

```
0.14.8.5.2 EMPTY_MODEL const Model smrat::EMPTY_MODEL = Model() [static]
```

## 0.14.9 smrat::analyzer Namespace Reference

### Data Structures

- struct [DegreeCollector](#)
- struct [AnalysisSettings](#)
- struct [AnalyzerStatistics](#)

### Functions

- template<typename P >  
void [collect\\_projection\\_size](#) (const std::string &prefix, const P &projection, [AnalyzerStatistics](#) &stats)
- template<typename Settings >  
void [perform\\_projection](#) (const std::string &prefix, const std::set< [ConstraintT](#) > &constraints, [AnalyzerStatistics](#) &stats)
- void [analyze\\_cad\\_projections](#) (const [FormulaT](#) &f, [AnalyzerStatistics](#) &stats)
- void [analyze\\_cnf](#) (const [FormulaT](#) &f, [AnalyzerStatistics](#) &stats)
- void [analyze\\_formula\\_types](#) (const [FormulaT](#) &f, [AnalyzerStatistics](#) &stats)
- void [analyze\\_variables](#) (const [FormulaT](#) &f, [AnalyzerStatistics](#) &stats)
- template<typename T >  
void [registerAnalyzerSettings](#) (T &parser)

### 0.14.9.1 Function Documentation

```
0.14.9.1.1 analyze_cad_projections() void smrat::analyzer::analyze_cad_projections (
 const FormulaT & f,
 AnalyzerStatistics & stats)
```

```
0.14.9.1.2 analyze_cnf() void smrat::analyzer::analyze_cnf (
 const FormulaT & f,
 AnalyzerStatistics & stats)
```

```
0.14.9.1.3 analyze_formula_types() void smtrat::analyzer::analyze_formula_types (
 const FormulaT & f,
 AnalyzerStatistics & stats)
```

```
0.14.9.1.4 analyze_variables() void smtrat::analyzer::analyze_variables (
 const FormulaT & f,
 AnalyzerStatistics & stats)
```

```
0.14.9.1.5 collect_projection_size() template<typename P >
void smtrat::analyzer::collect_projection_size (
 const std::string & prefix,
 const P & projection,
 AnalyzerStatistics & stats)
```

```
0.14.9.1.6 perform_projection() template<typename Settings >
void smtrat::analyzer::perform_projection (
 const std::string & prefix,
 const std::set< ConstraintT > & constraints,
 AnalyzerStatistics & stats)
```

```
0.14.9.1.7 registerAnalyzerSettings() template<typename T >
void smtrat::analyzer::registerAnalyzerSettings (
 T & parser)
```

## 0.14.10 smtrat::cad Namespace Reference

### Namespaces

- [debug](#)
- [full](#)
- [full\\_ec](#)
- [preprocessor](#)
- [projection](#)
- [projection\\_compare](#)
- [sample\\_compare](#)

*Contains comparison operators for samples and associated helpers.*

- [variable\\_ordering](#)

### Data Structures

- class [CAD](#)
- class [LiftingTree](#)
- class [Sample](#)
- struct [FullSampleComparator](#)
- struct [FullSampleComparator< Iterator, FullSampleCompareStrategy::Value >](#)
- struct [FullSampleComparator< Iterator, FullSampleCompareStrategy::Type >](#)
- struct [FullSampleComparator< Iterator, FullSampleCompareStrategy::T >](#)
- class [SampleIteratorQueue](#)
- class [BaseProjection](#)
- struct [PolynomialComparator](#)
- class [PolynomialLiftingQueue](#)
- class [Projection](#)

- class [ModelBasedProjection](#)
- class [Projection< Incrementality::FULL, Backtracking::HIDE, Settings >](#)
- class [Projection< Incrementality::FULL, BT, Settings >](#)
- class [ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >](#)

*This class implements a projection that supports no incrementality and expects backtracking to be in order.*
- class [Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >](#)

*This class implements a projection that supports no incrementality and expects backtracking to be in order.*
- class [Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >](#)

*This class implements a projection that supports no incrementality and allows backtracking to be out of order.*
- class [Projection< Incrementality::SIMPLE, BT, Settings >](#)
- class [ProjectionGlobalInformation](#)
- class [ProjectionLevelInformation](#)
- class [ProjectionPolynomialInformation](#)
- class [ProjectionInformation](#)
- struct [ProjectionOperator](#)
- struct [BaseSettings](#)
- class [CADConstraints](#)
- struct [CADCore](#)
- struct [CADCore< CoreHeuristic::BySample >](#)
- struct [CADCore< CoreHeuristic::PreferProjection >](#)
- struct [CADCore< CoreHeuristic::PreferSampling >](#)
- struct [CADCore< CoreHeuristic::Interleave >](#)
- struct [CADCore< CoreHeuristic::EnumerateAll >](#)
- struct [CADPreprocessorSettings](#)
- class [CADPreprocessor](#)
- class [ConflictGraph](#)

*Representation of a bipartite graph ( $C+S, E$ ) of vertices  $C$  and  $S$ , representing the constraints and samples, respectively.*
- class [MISGeneration](#)
- class [Origin](#)

*This class represents one or more origins of some object.*
- struct [PreprocessorSettings](#)
- class [Preprocessor](#)

## Typedefs

- using [RAN = smtrat::RAN](#)
- using [ConstraintSelection = carl::Bitset](#)
- using [OptionalID = std::optional< std::size\\_t >](#)
- using [Assignment = std::map< carl::Variable, RAN >](#)
- using [SampleLiftedWith = carl::Bitset](#)
- using [SampleRootOf = carl::Bitset](#)
- using [UPoly = carl::UnivariatePolynomial< Poly >](#)
- template<typename Settings >  
using [ProjectionT = Projection< Settings::incrementality, Settings::backtracking, Settings >](#)
- template<typename Settings >  
using [ModelBasedProjectionT = ModelBasedProjection< Settings::incrementality, Settings::backtracking, Settings >](#)

## Enumerations

- enum class `Incrementality` { `NONE` , `SIMPLE` , `FULL` }
- enum class `Backtracking` { `ORDERED` , `UNORDERED` , `HIDE` }
- enum class `ProjectionType` {
 `Collins` , `Hong` , `McCallum` , `McCallum_partial` ,
 `Lazard` , `Brown` }
- enum class `ProjectionCompareStrategy` {
 `D` , `PD` , `SD` , `ID` ,
 `LD` , `Default = ID` }
- enum class `SampleCompareStrategy` {
 `T` , `TLSA` , `TSA` , `TS` ,
 `LT` , `LTA` , `LTS` , `LTSA` ,
 `LS` , `S` , `Type` , `Value` ,
 `Default = LT` }
- enum class `FullSampleCompareStrategy` { `Type` , `Value` , `T` , `Default = T` }
- enum class `MISHeuristic` {
 `TRIVIAL` , `GREEDY` , `GREEDY_PRE` , `GREEDY_WEIGHTED` ,
 `EXACT` , `HYBRID` }
- enum class `CoreHeuristic` {
 `BySample` , `PreferProjection` , `PreferSampling` , `EnumerateAll` ,
 `Interleave` }

## Functions

- template<typename I , typename C >
 std::ostream & `operator<<` (std::ostream &os, const `SampleIteratorQueue`< I, C > &siq)
- template<typename PG >
 std::ostream & `operator<<` (std::ostream &os, const `PolynomialLiftingQueue`< PG > &plq)
- template<typename S >
 std::ostream & `operator<<` (std::ostream &os, const `Projection`< `Incrementality::FULL`, `Backtracking::HIDE` ,
 S > &p)
- std::ostream & `operator<<` (std::ostream &os, const `full::Polynomial` &p)
- template<typename S , Backtracking B>
 std::ostream & `operator<<` (std::ostream &os, const `Projection`< `Incrementality::FULL`, B, S > &p)
- template<typename S >
 std::ostream & `operator<<` (std::ostream &os, const `ModelBasedProjection`< `Incrementality::NONE` ,
 `Backtracking::ORDERED`, S > &p)
- template<typename S >
 std::ostream & `operator<<` (std::ostream &os, const `Projection`< `Incrementality::NONE`, `Backtracking::ORDERED` ,
 S > &p)
- template<typename S >
 std::ostream & `operator<<` (std::ostream &os, const `Projection`< `Incrementality::NONE`, `Backtracking::UNORDERED` ,
 S > &p)
- template<typename S , Backtracking B>
 std::ostream & `operator<<` (std::ostream &os, const `Projection`< `Incrementality::SIMPLE`, B, S > &p)
- template<typename Poly >
 `projection_compare::Candidate`< Poly > `candidate` (const Poly &p, const Poly &q, std::size\_t level)
  - std::ostream & `operator<<` (std::ostream &os, const `ProjectionGlobalInformation` &gi)
  - std::ostream & `operator<<` (std::ostream &os, const `ProjectionLevelInformation::LevelInfo` &li)
  - std::ostream & `operator<<` (std::ostream &os, const `ProjectionPolynomialInformation::PolyInfo` &pi)
- template<Backtracking BT>
 std::ostream & `operator<<` (std::ostream &os, const `CADConstraints`< BT > &cc)
- const auto & `settings_cadpp` ()
- std::ostream & `operator<<` (std::ostream &os, const `CADPreprocessor` &cadpp)
- std::ostream & `operator<<` (std::ostream &os, const `ConflictGraph` &cg)
- std::ostream & `operator<<` (std::ostream &os, const `Preprocessor` &cadpp)

### 0.14.10.1 Typedef Documentation

**0.14.10.1.1 Assignment** using `smtrat::cad::Assignment` = `typedef std::map<carl::Variable, RAN>`

**0.14.10.1.2 ConstraintSelection** using `smtrat::cad::ConstraintSelection` = `typedef carl::Bitset`

**0.14.10.1.3 ModelBasedProjectionT** template<typename Settings >  
using `smtrat::cad::ModelBasedProjectionT` = `typedef ModelBasedProjection<Settings::incrementality,`  
`Settings::backtracking, Settings>`

**0.14.10.1.4 OptionalID** using `smtrat::cad::OptionalID` = `typedef std::optional<std::size_t>`

**0.14.10.1.5 ProjectionT** template<typename Settings >  
using `smtrat::cad::ProjectionT` = `typedef Projection<Settings::incrementality, Settings::backtracking,`  
`Settings>`

**0.14.10.1.6 RAN** using `smtrat::cad::RAN` = `typedef smtrat::RAN`

**0.14.10.1.7 SampleLiftedWith** using `smtrat::cad::SampleLiftedWith` = `typedef carl::Bitset`

**0.14.10.1.8 SampleRootOf** using `smtrat::cad::SampleRootOf` = `typedef carl::Bitset`

**0.14.10.1.9 UPoly** using `smtrat::cad::UPoly` = `typedef carl::UnivariatePolynomial<Poly>`

### 0.14.10.2 Enumeration Type Documentation

**0.14.10.2.1 Backtracking** enum `smtrat::cad::Backtracking` [strong]

Enumerator

|           |  |
|-----------|--|
| ORDERED   |  |
| UNORDERED |  |
| HIDE      |  |

**0.14.10.2.2 CoreHeuristic** enum `smtrat::cad::CoreHeuristic` [strong]

Enumerator

|                  |  |
|------------------|--|
| BySample         |  |
| PreferProjection |  |
| PreferSampling   |  |

Enumerator

|              |  |
|--------------|--|
| EnumerateAll |  |
| Interleave   |  |

#### 0.14.10.2.3 FullSampleCompareStrategy enum `smtrat::cad::FullSampleCompareStrategy` [strong]

Enumerator

|         |  |
|---------|--|
| Type    |  |
| Value   |  |
| T       |  |
| Default |  |

#### 0.14.10.2.4 Incrementality enum `smtrat::cad::Incrementality` [strong]

Enumerator

|        |  |
|--------|--|
| NONE   |  |
| SIMPLE |  |
| FULL   |  |

#### 0.14.10.2.5 MISHeuristic enum `smtrat::cad::MISHeuristic` [strong]

Enumerator

|                 |  |
|-----------------|--|
| TRIVIAL         |  |
| GREEDY          |  |
| GREEDY_PRE      |  |
| GREEDY_WEIGHTED |  |
| EXACT           |  |
| HYBRID          |  |

#### 0.14.10.2.6 ProjectionCompareStrategy enum `smtrat::cad::ProjectionCompareStrategy` [strong]

Enumerator

|         |  |
|---------|--|
| D       |  |
| PD      |  |
| SD      |  |
| ID      |  |
| LD      |  |
| Default |  |

**0.14.10.2.7 ProjectionType** enum `smtrat::cad::ProjectionType` [strong]**Enumerator**

|                  |  |
|------------------|--|
| Collins          |  |
| Hong             |  |
| McCallum         |  |
| McCallum_partial |  |
| Lazard           |  |
| Brown            |  |

**0.14.10.2.8 SampleCompareStrategy** enum `smtrat::cad::SampleCompareStrategy` [strong]**Enumerator**

|         |  |
|---------|--|
| T       |  |
| TLSA    |  |
| TSA     |  |
| TS      |  |
| LT      |  |
| LTA     |  |
| LTS     |  |
| LTSA    |  |
| LS      |  |
| S       |  |
| Type    |  |
| Value   |  |
| Default |  |

**0.14.10.3 Function Documentation****0.14.10.3.1 candidate()** template<typename Poly >  
`projection_compare::Candidate<Poly>` smtrat::cad::candidate (const Poly & p, const Poly & q, std::size\_t level )**0.14.10.3.2 operator<<()** [1/16] template<Backtracking BT>  
std::ostream& smtrat::cad::operator<< (std::ostream & os, const CADConstraints< BT > & cc )**0.14.10.3.3 operator<<()** [2/16] std::ostream& smtrat::cad::operator<< (std::ostream & os, const CADPreprocessor & cadpp ) [inline]

- 0.14.10.3.4 operator<<()** [3/16] std::ostream & smtrat::cad::operator<< ( std::ostream & os, const ConflictGraph & cg )
- 0.14.10.3.5 operator<<()** [4/16] std::ostream& smtrat::cad::operator<< ( std::ostream & os, const full::Polynomial & p ) [inline]
- 0.14.10.3.6 operator<<()** [5/16] template<typename S > std::ostream& smtrat::cad::operator<< ( std::ostream & os, const ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, S > & p )
- 0.14.10.3.7 operator<<()** [6/16] template<typename PG > std::ostream& smtrat::cad::operator<< ( std::ostream & os, const PolynomialLiftingQueue< PG > & plq ) [inline]
- 0.14.10.3.8 operator<<()** [7/16] std::ostream& smtrat::cad::operator<< ( std::ostream & os, const Preprocessor & cadpp ) [inline]
- 0.14.10.3.9 operator<<()** [8/16] template<typename S , Backtracking B> std::ostream& smtrat::cad::operator<< ( std::ostream & os, const Projection< Incrementality::FULL, B, S > & p )
- 0.14.10.3.10 operator<<()** [9/16] template<typename S > std::ostream& smtrat::cad::operator<< ( std::ostream & os, const Projection< Incrementality::FULL, Backtracking::HIDE, S > & p )
- 0.14.10.3.11 operator<<()** [10/16] template<typename S > std::ostream& smtrat::cad::operator<< ( std::ostream & os, const Projection< Incrementality::NONE, Backtracking::ORDERED, S > & p )
- 0.14.10.3.12 operator<<()** [11/16] template<typename S > std::ostream& smtrat::cad::operator<< ( std::ostream & os, const Projection< Incrementality::NONE, Backtracking::UNORDERED, S > & p )
- 0.14.10.3.13 operator<<()** [12/16] template<typename S , Backtracking B> std::ostream& smtrat::cad::operator<< ( std::ostream & os, const Projection< Incrementality::SIMPLE, B, S > & p )

```
0.14.10.3.14 operator<<() [13/16] std::ostream& smtrat::cad::operator<< (
 std::ostream & os,
 const ProjectionGlobalInformation & gi) [inline]
```

```
0.14.10.3.15 operator<<() [14/16] std::ostream& smtrat::cad::operator<< (
 std::ostream & os,
 const ProjectionLevelInformation::LevelInfo & li) [inline]
```

```
0.14.10.3.16 operator<<() [15/16] std::ostream& smtrat::cad::operator<< (
 std::ostream & os,
 const ProjectionPolynomialInformation::PolyInfo & pi) [inline]
```

```
0.14.10.3.17 operator<<() [16/16] template<typename I , typename C >
std::ostream& smtrat::cad::operator<< (
 std::ostream & os,
 const SampleIteratorQueue< I, C > & siq) [inline]
```

```
0.14.10.3.18 settings_cadpp() const auto & smtrat::cad::settings_cadpp () [inline]
```

## 0.14.11 smtrat::cad::debug Namespace Reference

### Data Structures

- struct DotSubgraph
- struct IDSanitizer
- struct UnifiedData
- class TikzBasePrinter
- class TikzDAGPrinter
- class TikzTreePrinter
- class TikzHistoryPrinter

## 0.14.12 smtrat::cad::full Namespace Reference

### TypeDefs

- using Polynomial = std::optional< std::pair< UPoly, Origin > >

### 0.14.12.1 Typedef Documentation

```
0.14.12.1.1 Polynomial using smtrat::cad::full::Polynomial = typedef std::optional<std::pair<UPoly,Origin>>
```

## 0.14.13 smtrat::cad::full\_ec Namespace Reference

### TypeDefs

- using Polynomial = std::optional< UPoly >

### 0.14.13.1 Typedef Documentation

**0.14.13.1.1 Polynomial** using `smtrat::cad::full_ec::Polynomial` = `typedef std::optional<UPoly>`

## 0.14.14 smtrat::cad::preprocessor Namespace Reference

### Data Structures

- struct [Origins](#)
- class [AssignmentCollector](#)
- class [ResultantRule](#)
- struct [ConstraintUpdate](#)

### Functions

- `std::size_t complexity (const std::vector< FormulaT > &origin)`

### 0.14.14.1 Function Documentation

**0.14.14.1.1 complexity()** `std::size_t smtrat::cad::preprocessor::complexity ( const std::vector< FormulaT > & origin ) [inline]`

## 0.14.15 smtrat::cad::projection Namespace Reference

### Namespaces

- [brown](#)  
*Contains the implementation of Browns projection operator as specified in ? after Theorem 3.1.*
- [collins](#)  
*Contains the implementation of Collins projection operator as specified in ? below Theorem 4.*
- [hong](#)  
*Contains the implementation of Hongs projection operator as specified in ? in Section 2.2.*
- [lazard](#)  
*Contains the implementation of Lazards projection operator as specified in [Lauard1994].*
- [mccallum](#)  
*Contains the implementation of McCallums projection operator as specified in ?.*
- [mccallum\\_partial](#)  
*Contains the implementation of an optimized version McCallums projection operator.*

### Data Structures

- struct [Reducta](#)  
*Construct the set of reducta of the given polynomial.*

### Functions

- bool [canBeRemoved](#) (const [UPoly](#) &p)  
*Checks whether a polynomial can safely be ignored.*
- bool [canBeForwarded](#) (std::size\_t, const [UPoly](#) &p)  
*Checks whether a polynomial can safely be forwarded to the next level.*
- template<typename Poly >  
  bool [doesNotVanish](#) (const [Poly](#) &p)

*Tries to determine whether the given Poly vanishes for some assignment.*

- template<typename Poly >  
**Poly normalize** (const Poly &p)  
*Normalizes the given Poly by removing constant and duplicate factors.*
- template<typename Poly >  
**Poly resultant** (carl::Variable variable, const Poly &p, const Poly &q)  
*Computes the resultant of two polynomials.*
- template<typename Poly >  
**Poly discriminant** (carl::Variable variable, const Poly &p)  
*Computes the discriminant of a polynomial.*
- template<typename Poly >  
**std::vector< Poly > PSC** (const Poly &p, const Poly &q)  
*Computes the Principal Subresultant Coefficients of two polynomials.*
- template<typename Poly , typename Callback >  
**void returnPoly** (const Poly &p, Callback &&cb)

#### 0.14.15.1 Function Documentation

**0.14.15.1.1 canBeForwarded()** bool smtrat::cad::projection::canBeForwarded ( std::size\_t , const UPoly & p ) [inline]

Checks whether a polynomial can safely be forwarded to the next level.

**0.14.15.1.2 canBeRemoved()** bool smtrat::cad::projection::canBeRemoved ( const UPoly & p ) [inline]

Checks whether a polynomial can safely be ignored.

**0.14.15.1.3 discriminant()** template<typename Poly >  
**Poly** smtrat::cad::projection::discriminant ( carl::Variable variable, const Poly & p )

Computes the discriminant of a polynomial.

The polynomial is assumed to be univariate and thus knows its main variable. The given variable instead indicates the main variable of the resulting polynomial.

##### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>variable</i> | Main variable of the resulting polynomial. |
| <i>p</i>        | Input polynomial.                          |

##### Returns

Discriminant of p in main variable variable.

**0.14.15.1.4 doesNotVanish()** template<typename Poly >  
bool smtrat::cad::projection::doesNotVanish ( const Poly & p )

Tries to determine whether the given Poly vanishes for some assignment.

Returns true if the Poly is guaranteed not to vanish. Returns false otherwise.

```
0.14.15.1.5 normalize() template<typename Poly >
Poly smtrat::cad::projection::normalize (
 const Poly & p)
```

Normalizes the given Poly by removing constant and duplicate factors.

```
0.14.15.1.6 PSC() template<typename Poly >
std::vector<Poly> smtrat::cad::projection::PSC (
 const Poly & p,
 const Poly & q)
```

Computes the Principal Subresultant Coefficients of two polynomials.

```
0.14.15.1.7 resultant() template<typename Poly >
Poly smtrat::cad::projection::resultant (
 carl::Variable variable,
 const Poly & p,
 const Poly & q)
```

Computes the resultant of two polynomials.

The polynomials are assumed to be univariate polynomials and thus know their respective main variables. The given variable instead indicates the main variable of the resulting polynomial.

#### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>variable</i> | Main variable of the resulting polynomial. |
| <i>p</i>        | First input polynomial.                    |
| <i>q</i>        | Second input polynomial.                   |

#### Returns

Resultant of p and q in main variable variable.

```
0.14.15.1.8 returnPoly() template<typename Poly , typename Callback >
void smtrat::cad::projection::returnPoly (
 const Poly & p,
 Callback && cb)
```

## 0.14.16 smtrat::cad::projection::brown Namespace Reference

Contains the implementation of Browns projection operator as specified in ? after Theorem 3.1.

#### Functions

- template<typename Poly , typename Callback >  
void **single** (const Poly &p, carl::Variable variable, Callback &&cb)  
*Implements the part of Browns projection operator from ? that deals with a single polynomial  $p \leftarrow : \{leading\text{coeff}(p), discriminant(p)\}$ .*
- template<typename Poly , typename Callback >  
void **paired** (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)  
*Implements the part of Browns projection operator from ? that deals with two polynomials  $p$  and  $q$  which is just the respective part of McCallums projection operator [mccallum::paired](#).*

### 0.14.16.1 Detailed Description

Contains the implementation of Browns projection operator as specified in ? after Theorem 3.1.

### 0.14.16.2 Function Documentation

```
0.14.16.2.1 paired() template<typename Poly , typename Callback >
void smtrat::cad::projection::brown::paired (
 const Poly & p,
 const UPoly & q,
 carl::Variable variable,
 Callback && cb)
```

Implements the part of Browns projection operator from ? that deals with two polynomials  $p$  and  $q$  which is just the respective part of McCallums projection operator [mccallum::paired](#).

```
0.14.16.2.2 single() template<typename Poly , typename Callback >
void smtrat::cad::projection::brown::single (
 const Poly & p,
 carl::Variable variable,
 Callback && cb)
```

Implements the part of Browns projection operator from ? that deals with a single polynomial  $p \leftarrow : \{leadingcoeff(p), discriminant(p)\}$ .

## 0.14.17 smtrat::cad::projection::collins Namespace Reference

Contains the implementation of Collins projection operator as specified in ? below Theorem 4.

### Functions

- template<typename Poly , typename Callback >  
void **single** (const Poly &p, carl::Variable variable, Callback &&cb)  
*Implements the part of Collins projection operator from ? that deals with a single polynomial  $p \leftarrow : \bigcup_{B \in RED(p)} \{Idcf(B)\} \cup PSC(B, B')$ .*
- template<typename Poly , typename Callback >  
void **paired** (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)  
*Implements the part of Collins projection operator from ? that deals with two polynomials  $p$  and  $q \leftarrow : \bigcup_{B_1 \in RED(p), B_2 \in RED(q)} PSC(B_1, B_2)$ .*

### 0.14.17.1 Detailed Description

Contains the implementation of Collins projection operator as specified in ? below Theorem 4.

### 0.14.17.2 Function Documentation

```
0.14.17.2.1 paired() template<typename Poly , typename Callback >
void smtrat::cad::projection::collins::paired (
 const Poly & p,
 const UPoly & q,
 carl::Variable variable,
 Callback && cb)
```

Implements the part of Collins projection operator from ? that deals with two polynomials  $p$  and  $q \leftarrow : \bigcup_{B_1 \in RED(p), B_2 \in RED(q)} PSC(B_1, B_2)$ .

```
0.14.17.2.2 single() template<typename Poly , typename Callback >
void smtrat::cad::projection::collins::single (
 const Poly & p,
 carl::Variable variable,
 Callback && cb)
```

Implements the part of Collins projection operator from ? that deals with a single polynomial  $p \leftarrow \bigcup_{B \in RED(p)} \{ldcf(B)\} \cup PSC(B, B')$ .

## 0.14.18 smtrat::cad::projection::hong Namespace Reference

Contains the implementation of Hongs projection operator as specified in ? in Section 2.2.

### Functions

- template<typename Poly , typename Callback >  
void **single** (const Poly &p, carl::Variable variable, Callback &&cb)  
*Implements the part of Hongs projection operator from ? that deals with a single polynomial  $p$  which is just the respective part of Collins' projection operator collins::single.*
- template<typename Poly , typename Callback >  
void **paired** (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)  
*Implements the part of Hongs projection operator from ? that deals with two polynomials  $p$  and  $q \leftarrow \bigcup_{F \in RED(p)} PSC(F, q)$ .*

### 0.14.18.1 Detailed Description

Contains the implementation of Hongs projection operator as specified in ? in Section 2.2.

### 0.14.18.2 Function Documentation

```
0.14.18.2.1 paired() template<typename Poly , typename Callback >
void smtrat::cad::projection::hong::paired (
 const Poly & p,
 const UPoly & q,
 carl::Variable variable,
 Callback && cb)
```

Implements the part of Hongs projection operator from ? that deals with two polynomials  $p$  and  $q \leftarrow \bigcup_{F \in RED(p)} PSC(F, q)$ .

```
0.14.18.2.2 single() template<typename Poly , typename Callback >
void smtrat::cad::projection::hong::single (
 const Poly & p,
 carl::Variable variable,
 Callback && cb)
```

Implements the part of Hongs projection operator from ? that deals with a single polynomial  $p$  which is just the respective part of Collins' projection operator collins::single.

## 0.14.19 smtrat::cad::projection::lazard Namespace Reference

Contains the implementation of Lazards projection operator as specified in [Lauard1994].

### Functions

- template<typename Poly , typename Callback >  
void **single** (const Poly &p, carl::Variable variable, Callback &&cb)

Implements the part of Lazards projection operator from ? that deals with a single polynomial  $p \leftarrow : \{leading_{coeff}(p), trailing_{coeff}(p), discriminant(p)\}$ .

- template<typename Poly , typename Callback >  
void **paired** (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)  
*Implements the part of Lazard projection operator from ? that deals with two polynomials p and q which is just the respective part of McCallums projection operator [mccallum::paired](#).*

#### 0.14.19.1 Detailed Description

Contains the implementation of Lazards projection operator as specified in [\[Lauard1994\]](#).

#### 0.14.19.2 Function Documentation

**0.14.19.2.1 paired()** template<typename Poly , typename Callback >  
void smtrat::cad::projection::lazard::paired (  
    const Poly & p,  
    const UPoly & q,  
    carl::Variable variable,  
    Callback && cb )

Implements the part of Lazard projection operator from ? that deals with two polynomials p and q which is just the respective part of McCallums projection operator [mccallum::paired](#).

**0.14.19.2.2 single()** template<typename Poly , typename Callback >  
void smtrat::cad::projection::lazard::single (  
    const Poly & p,  
    carl::Variable variable,  
    Callback && cb )

Implements the part of Lazards projection operator from ? that deals with a single polynomial  $p \leftarrow : \{leading_{coeff}(p), trailing_{coeff}(p), discriminant(p)\}$ .

### 0.14.20 smtrat::cad::projection::mccallum Namespace Reference

Contains the implementation of McCallums projection operator as specified in ?.

#### Functions

- template<typename Poly , typename Callback >  
void **single** (const Poly &p, carl::Variable variable, Callback &&cb)  
*Implements the part of McCallums projection operator from ? that deals with a single polynomial  $p \leftarrow : coefficients(p) \cup \{discriminant(p)\}$ .*
- template<typename Poly , typename Callback >  
void **paired** (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)  
*Implements the part of McCallums projection operator from ? that deals with two polynomials p and q  $\leftarrow : \{resultant(p, q)\}$ .*

#### 0.14.20.1 Detailed Description

Contains the implementation of McCallums projection operator as specified in ?.

#### 0.14.20.2 Function Documentation

```
0.14.20.2.1 paired() template<typename Poly , typename Callback >
void smtrat::cad::projection::mccallum::paired (
 const Poly & p,
 const UPoly & q,
 carl::Variable variable,
 Callback && cb)
```

Implements the part of McCallums projection operator from ? that deals with two polynomials  $p$  and  $q \leftarrow : \{resultant(p, q)\}$ .

```
0.14.20.2.2 single() template<typename Poly , typename Callback >
void smtrat::cad::projection::mccallum::single (
 const Poly & p,
 carl::Variable variable,
 Callback && cb)
```

Implements the part of McCallums projection operator from ? that deals with a single polynomial  $p \leftarrow : coefficients(p) \cup \{discriminant(p)\}$ .

## 0.14.21 smtrat::cad::projection::mccallum\_partial Namespace Reference

Contains the implementation of an optimized version McCallums projection operator.

### Functions

- template<typename Poly , typename Callback >  
void **single** (const Poly &p, carl::Variable variable, Callback &&cb)
- template<typename Poly , typename Callback >  
void **paired** (const Poly &p, const UPoly &q, carl::Variable variable, Callback &&cb)

### 0.14.21.1 Detailed Description

Contains the implementation of an optimized version McCallums projection operator.

It is based on the observation by Brown that if some coefficient does not vanish only the leading coefficient is required.

### 0.14.21.2 Function Documentation

```
0.14.21.2.1 paired() template<typename Poly , typename Callback >
void smtrat::cad::projection::mccallum_partial::paired (
 const Poly & p,
 const UPoly & q,
 carl::Variable variable,
 Callback && cb)
```

```
0.14.21.2.2 single() template<typename Poly , typename Callback >
void smtrat::cad::projection::mccallum_partial::single (
 const Poly & p,
 carl::Variable variable,
 Callback && cb)
```

## 0.14.22 smtrat::cad::projection\_compare Namespace Reference

### Data Structures

- struct **level**

- struct `degree`
- struct `type`
- struct `ProjectionComparator_Impl`
- struct `ProjectionComparator`
- struct `ProjectionComparator< ProjectionCompareStrategy::D >`
- struct `ProjectionComparator< ProjectionCompareStrategy::PD >`
- struct `ProjectionComparator< ProjectionCompareStrategy::SD >`
- struct `ProjectionComparator< ProjectionCompareStrategy::ID >`
- struct `ProjectionComparator< ProjectionCompareStrategy::LD >`

## TypeDefs

- template<typename Poly >  
using `Candidate` = std::tuple< const Poly &, const Poly &, std::size\_t >
- using `lt` = std::less<>
- using `gt` = std::greater<>

## Functions

- template<typename Poly >  
std::ostream & `operator<<` (std::ostream &os, const `Candidate< Poly >` &c)
- template<typename Poly >  
auto `get` (const `Candidate< Poly >` &c, `level`)
- template<typename Poly >  
auto `get` (const `Candidate< Poly >` &c, `degree`)
- template<typename Poly >  
auto `get` (const `Candidate< Poly >` &c, `type`)
- template<typename Poly , typename tag , typename F >  
int `compareCriterion` (const `Candidate< Poly >` &lhs, const `Candidate< Poly >` &rhs, tag t, F &&f)  
*Compares the criterion given by t of two samples lhs and rhs using a comparator f.*
- template<typename Poly >  
bool `compare` (const `Candidate< Poly >` &lhs, const `Candidate< Poly >` &rhs)
- template<typename Poly , typename tag , typename F , typename... Tail>  
bool `compare` (const `Candidate< Poly >` &lhs, const `Candidate< Poly >` &rhs)

### 0.14.22.1 TypeDef Documentation

**0.14.22.1.1 Candidate** template<typename Poly >  
using `smtrat::cad::projection_compare::Candidate` = typedef std::tuple<const Poly&, const Poly&, std::size\_t>

**0.14.22.1.2 gt** using `smtrat::cad::projection_compare::gt` = typedef std::greater<>

**0.14.22.1.3 lt** using `smtrat::cad::projection_compare::lt` = typedef std::less<>

### 0.14.22.2 Function Documentation

**0.14.22.2.1 compare() [1/2]** template<typename Poly >  
bool `smtrat::cad::projection_compare::compare` (  
    const `Candidate< Poly >` & lhs,  
    const `Candidate< Poly >` & rhs )

```
0.14.22.2.2 compare() [2/2] template<typename Poly , typename tag , typename F , typename... Tail>
bool smtrat::cad::projection_compare::compare (
 const Candidate< Poly > & lhs,
 const Candidate< Poly > & rhs)
```

```
0.14.22.2.3 compareCriterion() template<typename Poly , typename tag , typename F >
int smtrat::cad::projection_compare::compareCriterion (
 const Candidate< Poly > & lhs,
 const Candidate< Poly > & rhs,
 tag t,
 F && f)
```

Compares the criterion given by `t` of two samples `lhs` and `rhs` using a comparator `f`.

Returns 0 if the values are the same, -1 if `lhs` should be used first and 1 if `rhs` should be used first.

```
0.14.22.2.4 get() [1/3] template<typename Poly >
auto smtrat::cad::projection_compare::get (
 const Candidate< Poly > & c,
 degree)
```

```
0.14.22.2.5 get() [2/3] template<typename Poly >
auto smtrat::cad::projection_compare::get (
 const Candidate< Poly > & c,
 level)
```

```
0.14.22.2.6 get() [3/3] template<typename Poly >
auto smtrat::cad::projection_compare::get (
 const Candidate< Poly > & c,
 type)
```

```
0.14.22.2.7 operator<<() template<typename Poly >
std::ostream& smtrat::cad::projection_compare::operator<< (
 std::ostream & os,
 const Candidate< Poly > & c)
```

## 0.14.23 smtrat::cad::sample\_compare Namespace Reference

Contains comparison operators for samples and associated helpers.

### Data Structures

- struct `level`
- struct `size`
- struct `absvalue`
- struct `type`
- struct `SampleComparator_Impl`
- struct `SampleComparator`
- struct `SampleComparator< Iterator, SampleCompareStrategy::Value >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::Type >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::T >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::TLSA >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::TSA >`

- struct `SampleComparator< Iterator, SampleCompareStrategy::TS >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::LT >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::LTA >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::LTS >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::LTSA >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::LS >`
- struct `SampleComparator< Iterator, SampleCompareStrategy::S >`

## Typedefs

- using `lt = std::less<>`
- using `gt = std::greater<>`

## Functions

- template<typename It>  
auto `get` (const It &it, `level`)
- template<typename It>  
auto `get` (const It &it, `size`)
- template<typename It>  
auto `get` (const It &it, `absvalue`)
- template<typename It>  
auto `get` (const It &it, `type`)
- template<typename It, typename tag, typename F>  
int `compareCriterion` (const It &lhs, const It &rhs, tag t, F &&f)  
*Compares the criterion given by t of two samples lhs and rhs using a comparator f.*
- template<typename It>  
bool `compare` (const It &lhs, const It &rhs)
- template<typename It, typename tag, typename F, typename... Tail>  
bool `compare` (const It &lhs, const It &rhs)

### 0.14.23.1 Detailed Description

Contains comparison operators for samples and associated helpers.

The comparators implement a less-than operators and large samples are considered first. Hence if `lhs` has greater priority than `rhs`, the result should be false.

### 0.14.23.2 Typedef Documentation

#### 0.14.23.2.1 `gt` using `smtrat::cad::sample_compare::gt` = `typedef std::greater<>`

#### 0.14.23.2.2 `lt` using `smtrat::cad::sample_compare::lt` = `typedef std::less<>`

### 0.14.23.3 Function Documentation

#### 0.14.23.3.1 `compare()` [1/2] template<typename It>

```
bool smtrat::cad::sample_compare::compare (
 const It & lhs,
 const It & rhs)
```

```
0.14.23.3.2 compare() [2/2] template<typename It , typename tag , typename F , typename...
Tail>
bool smtrat::cad::sample_compare::compare (
 const It & lhs,
 const It & rhs)
```

```
0.14.23.3.3 compareCriterion() template<typename It , typename tag , typename F >
int smtrat::cad::sample_compare::compareCriterion (
 const It & lhs,
 const It & rhs,
 tag t,
 F && f)
```

Compares the criterion given by `t` of two samples `lhs` and `rhs` using a comparator `f`.

Returns 0 if the values are the same, -1 if `lhs` should be used first and 1 if `rhs` should be used first.

```
0.14.23.3.4 get() [1/4] template<typename It >
auto smtrat::cad::sample_compare::get (
 const It & it,
 absvalue)
```

```
0.14.23.3.5 get() [2/4] template<typename It >
auto smtrat::cad::sample_compare::get (
 const It & it,
 level)
```

```
0.14.23.3.6 get() [3/4] template<typename It >
auto smtrat::cad::sample_compare::get (
 const It & it,
 size)
```

```
0.14.23.3.7 get() [4/4] template<typename It >
auto smtrat::cad::sample_compare::get (
 const It & it,
 type)
```

## 0.14.24 smtrat::cad::variable\_ordering Namespace Reference

### Functions

- std::vector< carl::Variable > [triangular\\_ordering](#) (const std::vector< Poly > &polys)

#### 0.14.24.1 Function Documentation

```
0.14.24.1.1 triangular_ordering() std::vector< carl::Variable > smtrat::cad::variable_ordering::
::triangular_ordering (
 const std::vector< Poly > & polys)
```

## 0.14.25 smtrat::cadcells Namespace Reference

A framework for sample-based CAD algorithms.

## Namespaces

- [algorithms](#)  
*Various algorithms as well as helper functions for developing new algorithms.*
- [datastructures](#)  
*Main datastructures.*
- [helper](#)
- [operators](#)  
*Projection operators.*
- [representation](#)  
*Heuristics for computing representations.*

## Typedefs

- using [VariableOrdering](#) = std::vector< carl::Variable >
- using [Polynomial](#) = carl::ContextPolynomial< [Rational](#) >
- using [Constraint](#) = carl::BasicConstraint< [Polynomial](#) >
- using [MultivariateRoot](#) = carl::MultivariateRoot< [Polynomial](#) >
- using [VariableComparison](#) = carl::VariableComparison< [Polynomial](#) >
- using [Atom](#) = std::variant< [Constraint](#), [VariableComparison](#) >
- using [RAN](#) = Polynomial::RootType
- using [Assignment](#) = carl::Assignment< [RAN](#) >

## Variables

- static const [Assignment](#) `empty_assignment`

### 0.14.25.1 Detailed Description

A framework for sample-based CAD algorithms.

This is a highly modular framework for sample-based CAD algorithms, i.e. single cell construction and coverings. The basic idea is to have properties (of some polynomials or real root functions); each property has a level w.r.t. to a variable ordering (i.e. the index of the main variable of a polynomial) and rules operating on them, each replacing a property by a "simpler" set of properties (eventually reducing the level). At some stage, we delineate properties (that is, ordering the root under a partial sample), determine an ordering and cell boundaries (called representation) to continue proving properties. For more details on the general framework, we refer to the paper.

The structure of this implementation is as follows:

- [cadcells::datastructures](#) contains the main datastructures. Read here for more details on the general structure of the framework.
- [cadcells::operators](#) defines the properties, the rules and methods to delineate properties. These are used by operators which provide an interface for performing projections on certain steps.
- [cadcells::representation](#) provides heuristics for computing the representations for cells, coverings and delineations.
- [cadcells::algorithms](#) contains helper methods, building blocks for algorithms as well as algorithms themselves. Go here for usage of the framework and high-level interfaces.

#### 0.14.25.1.1 Quick start

For an introduction, we refer to the code of `algorithms::oncell`.

### 0.14.25.2 Typedef Documentation

#### 0.14.25.2.1 Assignment

```
using smtrat::cadcells::Assignment = typedef carl::Assignment<RAN>
```

**0.14.25.2.2 Atom** using `smtrat::cadcells::Atom = std::variant<Constraint, VariableComparison>`

**0.14.25.2.3 Constraint** using `smtrat::cadcells::Constraint = carl::BasicConstraint<Polynomial>`

**0.14.25.2.4 MultivariateRoot** using `smtrat::cadcells::MultivariateRoot = carl::MultivariateRoot<Polynomial>`

**0.14.25.2.5 Polynomial** using `smtrat::cadcells::Polynomial = carl::ContextPolynomial<Rational>`

**0.14.25.2.6 RAN** using `smtrat::cadcells::RAN = carl::Polynomial::RootType`

**0.14.25.2.7 VariableComparison** using `smtrat::cadcells::VariableComparison = carl::VariableComparison<Polynomial>`

**0.14.25.2.8 VariableOrdering** using `smtrat::cadcells::VariableOrdering = std::vector<carl::Variable>`

### 0.14.25.3 Variable Documentation

**0.14.25.3.1 empty\_assignment** const `Assignment smtrat::cadcells::empty_assignment [static]`

## 0.14.26 smtrat::cadcells::algorithms Namespace Reference

Various algorithms as well as helper functions for developing new algorithms.

### Functions

- template<`cadcells::operators::op op, representation::CellHeuristic cell_heuristic>`  
`std::optional< std::pair< carl::Variable, datastructures::SymbolicInterval > > get_interval (datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > &cell_deriv)`  
*Single cell construction algorithm.*
- template<`cadcells::operators::op op, representation::CoveringHeuristic covering_heuristic>`  
`std::optional< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_level_covering (datastructures::Projections &proj, const std::vector< Atom > &constraints, const Assignment &sample)`  
*Computes an unsat covering w.r.t.*
- template<`cadcells::operators::op op>`  
`std::vector< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_unsat_intervals (const Constraint &c, datastructures::Projections &proj, const Assignment &sample)`
- template<`cadcells::operators::op op>`  
`std::vector< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_unsat_intervals (const VariableComparison &c, datastructures::Projections &proj, const Assignment &sample)`
- template<`cadcells::operators::op op>`  
`std::vector< datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >::type > > get_unsat_intervals (const Atom &c, datastructures::Projections &proj, const Assignment &sample)`  
*Returns the unsat intervals of the given atom w.r.t.*

### 0.14.26.1 Detailed Description

Various algorithms as well as helper functions for developing new algorithms.

### 0.14.26.2 Function Documentation

**0.14.26.2.1 get\_interval()** template<cadcells::operators::op op, representation::CellHeuristic cell\_heuristic>  
`std::optional<std::pair<carl::Variable, datastructures::SymbolicInterval> > smrat::cadcells->`  
`::algorithms::get_interval (`  
 `datastructures::SampledDerivationRef< typename operators::PropertiesSet< op >>->`  
`::type > & cell_deriv )`  
Single cell construction algorithm.

#### Template Parameters

|                       |                     |
|-----------------------|---------------------|
| <i>op</i>             | The operator.       |
| <i>cell_heuristic</i> | The cell heuristic. |

#### Parameters

|                   |                                                 |
|-------------------|-------------------------------------------------|
| <i>cell_deriv</i> | A derivation object to construct the cell from. |
|-------------------|-------------------------------------------------|

#### Returns

A vector of pairs of variables and their symbolic intervals on success or std::nullopt otherwise.

**0.14.26.2.2 get\_level\_covering()** template<cadcells::operators::op op, representation::Covering->  
Heuristic covering\_heuristic>  
`std::optional<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>>->`  
`::type > smrat::cadcells::algorithms::get_level_covering (`  
 `datastructures::Projections & proj,`  
 `const std::vector< Atom > & constraints,`  
 `const Assignment & sample )`

Computes an unsat covering w.r.t.  
a set of constraints.

If a constraint is univariate under a sample, for the unassigned variables, the constraint induces intervals in which the constraint will be conflicting. If multiple such intervals from a set of constraints cover the real line, then the partial sample cannot be extended without making a conflict. This conflict is generalized.

#### Parameters

|                    |                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>constraints</i> | Atoms of the same level such that sample cannot be extended for the highest variable without making one atom false. Note that this property is not checked. |
| <i>sample</i>      | A sample such that all but the highest variable in constraints are assigned.                                                                                |

#### Returns

A sampled derivation which contains the information to reproduce the conflict.

**0.14.26.2.3 get\_unsat\_intervals()** [1/3] template<cadcells::operators::op op>

```
std::vector<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>><
::type> > smtrat::cadcells::algorithms::get_unsat_intervals (
 const Atom & c,
 datastructures::Projections & proj,
 const Assignment & sample)
```

Returns the unsat intervals of the given atom w.r.t.  
an underlying sample.

#### Parameters

|               |                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>c</i>      | A constraint or a variable comparison.                                                                                                 |
| <i>sample</i> | A sample point such that the highest variable in <i>c</i> w.r.t. the variable ordering in <i>proj</i> is the only unassigned variable. |

#### Returns

A list of sampled derivations with the same delineated derivations. The samples for the unassigned variables are sampled from the respective interval.

#### 0.14.26.2.4 `get_unsat_intervals()` [2/3] template<cadcells::operators::op op>

```
std::vector<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>><
::type> > smtrat::cadcells::algorithms::get_unsat_intervals (
 const Constraint & c,
 datastructures::Projections & proj,
 const Assignment & sample)
```

#### 0.14.26.2.5 `get_unsat_intervals()` [3/3] template<cadcells::operators::op op>

```
std::vector<datastructures::SampledDerivationRef<typename operators::PropertiesSet<op>><
::type> > smtrat::cadcells::algorithms::get_unsat_intervals (
 const VariableComparison & c,
 datastructures::Projections & proj,
 const Assignment & sample)
```

## 0.14.27 smtrat::cadcells::datastructures Namespace Reference

Main datastructures.

### Namespaces

- [detail](#)

### Data Structures

- struct [TaggedIndexedRoot](#)
- class [DelineationInterval](#)

*An interval of a delineation.*

- class [Delineation](#)

*Represents the delineation of a set of polynomials (at a sample), that is.*

- class [BaseDerivation](#)

*A [BaseDerivation](#) has a level and a set of properties of this level, and an underlying derivation representing the lower levels.*

- class [DelineatedDerivation](#)

*A [DelineatedDerivation](#) is a [BaseDerivation](#) with a [Delineation](#) and an underlying [SampledDerivation](#).*

- class [SampledDerivation](#)

A [SampledDerivation](#) is a [DelineatedDerivation](#) with a sample and an [DelineationInterval](#) w.r.t.

- class [DerivationRef](#)  
*A reference to a derivation, which is either.*
- struct [PolyRef](#)  
*Refers to a polynomial.*
- class [PolyPool](#)  
*A pool for polynomials.*
- class [Projections](#)  
*Encapsulates all computations on polynomials.*
- struct [property\\_hash](#)
- struct [PropertiesTContent](#)
- struct [PropertiesTContent< T, true >](#)
- struct [PropertiesTContent< T, false >](#)
- struct [PropertiesT](#)  
*Set of properties.*
- struct [PropertiesT< T, Ts... >](#)
- struct [CellRepresentation](#)  
*Represents a cell.*
- struct [CoveringRepresentation](#)  
*Represents a covering over a cell.*
- struct [IndexedRoot](#)  
*Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample).*
- struct [CompoundMin](#)  
*Represents the minimum function of the contained indexed root functions.*
- struct [CompoundMax](#)  
*Represents the maximum function of the contained indexed root functions.*
- class [RootFunction](#)
- class [Bound](#)  
*Bound type of a [SymbolicInterval](#).*
- class [SymbolicInterval](#)  
*A symbolic intreal whose bounds are defined by indexed root expressions.*
- class [CoveringDescription](#)  
*Describes a covering of the real line by [SymbolicIntervals](#) (given an appropriate sample).*
- struct [IndexedRootRelation](#)  
*A relation between two roots.*
- class [IndexedRootOrdering](#)  
*Describes an ordering of [IndexedRoots](#).*

## Typedefs

- using [RootMap](#) = std::map< [RAN](#), std::vector< [TaggedIndexedRoot](#) > >
- using [RootMapPlain](#) = std::map< [RAN](#), std::vector< [IndexedRoot](#) > >
- template<typename Properties>  
using [BaseDerivationRef](#) = std::shared\_ptr< [BaseDerivation< Properties](#) > >
- template<typename Properties>  
using [DelineatedDerivationRef](#) = std::shared\_ptr< [DelineatedDerivation< Properties](#) > >
- template<typename Properties>  
using [SampledDerivationRef](#) = std::shared\_ptr< [SampledDerivation< Properties](#) > >
- template<typename T>  
using [PropertiesTSet](#) = boost::container::flat\_set< T >

## Functions

- std::ostream & **operator<<** (std::ostream &os, const TaggedIndexedRoot &data)
- bool **operator==** (const TaggedIndexedRoot &lhs, const TaggedIndexedRoot &rhs)
- bool **operator<** (const TaggedIndexedRoot &lhs, const TaggedIndexedRoot &rhs)
- std::ostream & **operator<<** (std::ostream &os, const DelineationInterval &data)
- std::ostream & **operator<<** (std::ostream &os, const Delineation &data)
- bool **lower\_lt\_lower** (const DelineationInterval &del1, const DelineationInterval &del2)
 

*Compares the lower bounds of two DelineationIntervals.*
- bool **lower\_eq\_lower** (const DelineationInterval &del1, const DelineationInterval &del2)
 

*Compares the lower bounds of two DelineationIntervals.*
- bool **upper\_lt\_upper** (const DelineationInterval &del1, const DelineationInterval &del2)
 

*Compares the upper bounds of two DelineationIntervals.*
- bool **upper\_lt\_lower** (const DelineationInterval &del1, const DelineationInterval &del2)
 

*Compares an upper bound with a lower bound of DelineationIntervals.*
- template<typename P>
   
bool **operator==** (const DerivationRef< P > &lhs, const DerivationRef< P > &rhs)
- template<typename P>
   
bool **operator<** (const DerivationRef< P > &lhs, const DerivationRef< P > &rhs)
- template<typename Properties>
   
**DerivationRef< Properties > make\_derivation** (Projections &proj, const Assignment &assignment, size\_t level)
 

*Initializes a derivation according the the given assignment and level.*
- template<typename Properties>
   
**SampledDerivationRef< Properties > make\_sampled\_derivation** (DelineatedDerivationRef< Properties > delineated, const RAN &main\_sample)
 

*Initializes a sampled derivation w.r.t.*
- template<typename Properties>
   
void **merge\_underlying** (std::vector< SampledDerivationRef< Properties > > &derivations)
 

*Merges the underlying derivations of a set of sampled derivations.*
- bool **operator<** (const PolyRef &lhs, const PolyRef &rhs)
- bool **operator==** (const PolyRef &lhs, const PolyRef &rhs)
- bool **operator!=** (const PolyRef &lhs, const PolyRef &rhs)
- std::ostream & **operator<<** (std::ostream &os, const PolyRef &data)
- template<class T, class... Ts>
   
void **prop\_insert** (PropertiesT< T, Ts... > &sets, const T &element)
- template<class S, class T, class... Ts, typename std::enable\_if< !std::is\_same< S, T >::value >::type>
   
void **prop\_insert** (PropertiesT< T, Ts... > &sets, const S &element)
- template<class T, class... Ts>
   
bool **prop\_has** (const PropertiesT< T, Ts... > &sets, const T &element)
- template<class S, class T, class... Ts, typename std::enable\_if< !std::is\_same< S, T >::value >::type>
   
bool **prop\_has** (const PropertiesT< T, Ts... > &sets, const S &element)
- template<class T, class... Ts>
   
const auto & **prop\_get** (const PropertiesT< T, Ts... > &sets)
- template<class S, class T, class... Ts, typename std::enable\_if< !std::is\_same< S, T >::value >::type>
   
const auto & **prop\_get** (const PropertiesT< T, Ts... > &sets)
- template<class T, class... Ts>
   
void **merge** (PropertiesT< T, Ts... > &sets\_a, const PropertiesT< T, Ts... > &sets\_b)
- template<typename P>
   
std::ostream & **operator<<** (std::ostream &os, const CellRepresentation< P > &data)
- template<typename P>
   
std::ostream & **operator<<** (std::ostream &os, const CoveringRepresentation< P > &data)
- bool **operator==** (const IndexedRoot &lhs, const IndexedRoot &rhs)
- bool **operator<** (const IndexedRoot &lhs, const IndexedRoot &rhs)
- bool **operator!=** (const IndexedRoot &lhs, const IndexedRoot &rhs)

- std::ostream & `operator<<` (std::ostream &os, const `IndexedRoot` &data)
- bool `operator==` (const `CompoundMin` &lhs, const `CompoundMin` &rhs)
- bool `operator<` (const `CompoundMin` &lhs, const `CompoundMin` &rhs)
- bool `operator!=` (const `CompoundMin` &lhs, const `CompoundMin` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `CompoundMin` &data)
- bool `operator==` (const `CompoundMax` &lhs, const `CompoundMax` &rhs)
- bool `operator<` (const `CompoundMax` &lhs, const `CompoundMax` &rhs)
- bool `operator!=` (const `CompoundMax` &lhs, const `CompoundMax` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `CompoundMax` &data)
- bool `operator==` (const `RootFunction` &lhs, const `RootFunction` &rhs)
- bool `operator<` (const `RootFunction` &lhs, const `RootFunction` &rhs)
- bool `operator!=` (const `RootFunction` &lhs, const `RootFunction` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `RootFunction` &data)
- std::ostream & `operator<<` (std::ostream &os, const `SymbolicInterval` &data)
- std::ostream & `operator<<` (std::ostream &os, const `CoveringDescription` &data)
- bool `operator==` (const `IndexedRootRelation` &lhs, const `IndexedRootRelation` &rhs)
- bool `operator<` (const `IndexedRootRelation` &lhs, const `IndexedRootRelation` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `IndexedRootRelation` &data)
- std::ostream & `operator<<` (std::ostream &os, const `IndexedRootOrdering` &data)

#### 0.14.27.1 Detailed Description

Main datastructures.

**0.14.27.1.1 Polynomials and projection results** We assume a fixed variable ordering `VariableOrdering`. All polynomials are pooled in `PolyPool` w.r.t. this ordering, that is, they are identified by a `PolyRef` (a pair of the polynomial's level and an ID on this level). Projection results are cached in `Projections`. The latter holds a reference to a `PolyPool`. An instance `Projections` will be required for all kinds of operations and passed during various function calls. Thus, we initialize these data structures as follows:

```
VariableOrdering vars; // a variable ordering of all variables that will occur.
datastructures::PolyPool pool(vars);
datastructures::Projections proj(pool);
```

Note that `PolyPool` and `Projections` do not do reference counting; the cache needs to be cleared explicitly, see `Projections::clear_cache` and `Projections::clear_assignment_cache`.

**0.14.27.1.2 Basic datastructures** The basic datastructures for representing mathematical objects are `datastructures::IndexedRoot`, `datastructures::SymbolicInterval`, `datastructures::CoveringDescription`, and `datastructures::IndexedRootOrdering`.

**0.14.27.1.3 Properties** `datastructures::PropertiesT` is the datastructure for storing properties on a single level. If you work on an algorithm using this framework, you don't need to know this datastructure. If you need to implement a new property, then you find the interfaces a property needs to implement there.

**0.14.27.1.4 Delineations** `datastructures::Delineation` is the datastructure for storing the delineation of the indexed roots that stem from properties.

If you work on an algorithm using this framework, you don't need to know this datastructure. If you need to implement a new property or a heuristic for describing a cell (i.e. compute a representation), then you might be interested in this.

**0.14.27.1.5 Derivations** In the derivation datastructures, all data regarding the derivation is stored - that is the current sample points, the set of properties, the delineation of roots and the possible cell boundaries.

This datastructure is recursive, that is, each derivation object handles only a single level and holds a pointer to a derivation object of the next lower level (the underlying derivation). All operations are forwarded to lower levels whenever necessary.

Furthermore, there are three types of derivations:

- `datastructures::BaseDerivation` assumes no sample point or whatever. It simply stores properties.

- `datastructures::DelineatedDerivation` extends the `datastructures::BaseDerivation`; it requires that the underlying derivation is a `datastructures::SampledDerivation`, thus a sample for all lower levels is present under which the properties of the current level can be delineated (i.e. their zeros can be isolated and sorted).
- `datastructures::SampledDerivation` extends the `datastructures::DelineatedDerivation`, thus a full sample point is available for which a cell can be isolated. The derivation datastructures are thus two-dimensional recursive. Note that multiple derivation objects can have common underlying cells or even share the same base or delineated derivation.

`datastructures::DerivationRef` allows to reference a derivation independent of its type.

For initializing derivations properly, use `datastructures::make_derivation`. To create a sampled derivation from a delineated derivation and a sample, use `datastructures::make_sampled_derivation`. For merging underlying derivations, use `datastructures::merge_underlying`.

For more information on memory management read `datastructures::DerivationRef`.

**0.14.27.1.6 Representations** The derivation objects do not store heuristic decisions, they just describe the current state. At some point, a representation of this state needs to be determined that is passed to the operator. This heuristic decision is stored in `datastructures::CellRepresentation` or `datastructures::CoveringRepresentation`. The heuristics for computing representations are in `representation`.

**0.14.27.1.7 Operators** The `operators` work on derivations and representations.

## 0.14.27.2 Typedef Documentation

### 0.14.27.2.1 BaseDerivationRef template<typename Properties >

```
using smtrat::cadcells::datastructures::BaseDerivationRef = typedef std::shared_ptr<BaseDerivation<Properties>>
```

### 0.14.27.2.2 DelineatedDerivationRef template<typename Properties >

```
using smtrat::cadcells::datastructures::DelineatedDerivationRef = typedef std::shared_ptr<DelineatedDerivation<Properties>>
```

### 0.14.27.2.3 PropertiesTSet template<typename T >

```
using smtrat::cadcells::datastructures::PropertiesTSet = typedef boost::container::flat_set<T>
```

### 0.14.27.2.4 RootMap using smtrat::cadcells::datastructures::RootMap = typedef std::map<RAN, std::vector<TaggedIndexedRoot>>

### 0.14.27.2.5 RootMapPlain using smtrat::cadcells::datastructures::RootMapPlain = typedef std::map<RAN, std::vector<IndexedRoot>>

### 0.14.27.2.6 SampledDerivationRef template<typename Properties >

```
using smtrat::cadcells::datastructures::SampledDerivationRef = typedef std::shared_ptr<SampledDerivation<Properties>>
```

## 0.14.27.3 Function Documentation

**0.14.27.3.1 `lower_eq_lower()`** `bool smtrat::cadcells::datastructures::lower_eq_lower (`  
    `const DelineationInterval & del1,`  
    `const DelineationInterval & del2 ) [inline]`

Compares the lower bounds of two DelineationIntervals.

It respects whether the interval is a section or sector.

**0.14.27.3.2 `lower_lt_lower()`** `bool smtrat::cadcells::datastructures::lower_lt_lower (`  
    `const DelineationInterval & del1,`  
    `const DelineationInterval & del2 ) [inline]`

Compares the lower bounds of two DelineationIntervals.

It respects whether the interval is a section or sector.

**0.14.27.3.3 `make_derivation()`** `template<typename Properties >`  
`DerivationRef<Properties> smtrat::cadcells::datastructures::make_derivation (`  
    `Projections & proj,`  
    `const Assignment & assignment,`  
    `size_t level )`

Initializes a derivation according the the given assignment and level.

**0.14.27.3.4 `make_sampled_derivation()`** `template<typename Properties >`  
`SampledDerivationRef<Properties> smtrat::cadcells::datastructures::make_sampled_derivation (`  
    `DelineatedDerivationRef<Properties> delineated,`  
    `const RAN & main_sample )`

Initializes a sampled derivation w.r.t.

the delineated derivation and sample.

**0.14.27.3.5 `merge()`** `template<class T , class... Ts>`  
`void smtrat::cadcells::datastructures::merge (`  
    `PropertiesT< T, Ts... > & sets_a,`  
    `const PropertiesT< T, Ts... > & sets_b )`

**0.14.27.3.6 `merge_underlying()`** `template<typename Properties >`

`void smtrat::cadcells::datastructures::merge_underlying (`  
    `std::vector< SampledDerivationRef<Properties>> & derivations )`

Merges the underlying derivations of a set of sampled derivations.

After the operation, all sampled derivations point to the same underlying derivation.

**0.14.27.3.7 `operator"!=() [1/5]`** `bool smtrat::cadcells::datastructures::operator!= (`  
    `const CompoundMax & lhs,`  
    `const CompoundMax & rhs ) [inline]`

**0.14.27.3.8 `operator"!=() [2/5]`** `bool smtrat::cadcells::datastructures::operator!= (`  
    `const CompoundMin & lhs,`  
    `const CompoundMin & rhs ) [inline]`

**0.14.27.3.9 `operator"!=() [3/5]`** `bool smtrat::cadcells::datastructures::operator!= (`  
    `const IndexedRoot & lhs,`  
    `const IndexedRoot & rhs ) [inline]`

**0.14.27.3.10 operator"!=() [4/5]** bool smtrat::cadcells::datastructures::operator!= ( const PolyRef & lhs, const PolyRef & rhs ) [inline]

**0.14.27.3.11 operator"!=() [5/5]** bool smtrat::cadcells::datastructures::operator!= ( const RootFunction & lhs, const RootFunction & rhs ) [inline]

**0.14.27.3.12 operator<() [1/8]** bool smtrat::cadcells::datastructures::operator< ( const CompoundMax & lhs, const CompoundMax & rhs ) [inline]

**0.14.27.3.13 operator<() [2/8]** bool smtrat::cadcells::datastructures::operator< ( const CompoundMin & lhs, const CompoundMin & rhs ) [inline]

**0.14.27.3.14 operator<() [3/8]** template<typename P> bool smtrat::cadcells::datastructures::operator< ( const DerivationRef< P > & lhs, const DerivationRef< P > & rhs )

**0.14.27.3.15 operator<() [4/8]** bool smtrat::cadcells::datastructures::operator< ( const IndexedRoot & lhs, const IndexedRoot & rhs ) [inline]

**0.14.27.3.16 operator<() [5/8]** bool smtrat::cadcells::datastructures::operator< ( const IndexedRootRelation & lhs, const IndexedRootRelation & rhs ) [inline]

**0.14.27.3.17 operator<() [6/8]** bool smtrat::cadcells::datastructures::operator< ( const PolyRef & lhs, const PolyRef & rhs ) [inline]

**0.14.27.3.18 operator<() [7/8]** bool smtrat::cadcells::datastructures::operator< ( const RootFunction & lhs, const RootFunction & rhs ) [inline]

**0.14.27.3.19 operator<() [8/8]** bool smtrat::cadcells::datastructures::operator< ( const TaggedIndexedRoot & lhs, const TaggedIndexedRoot & rhs ) [inline]

**0.14.27.3.20 operator<<() [1/14]** template<typename P> std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const CellRepresentation< P > & data ) [inline]

- 0.14.27.3.21** `operator<<()` [2/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const CompoundMax & data ) [inline]`
- 0.14.27.3.22** `operator<<()` [3/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const CompoundMin & data ) [inline]`
- 0.14.27.3.23** `operator<<()` [4/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const CoveringDescription & data ) [inline]`
- 0.14.27.3.24** `operator<<()` [5/14] `template<typename P > std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const CoveringRepresentation< P > & data )`
- 0.14.27.3.25** `operator<<()` [6/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const Delineation & data ) [inline]`
- 0.14.27.3.26** `operator<<()` [7/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const DelineationInterval & data ) [inline]`
- 0.14.27.3.27** `operator<<()` [8/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const IndexedRoot & data ) [inline]`
- 0.14.27.3.28** `operator<<()` [9/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const IndexedRootOrdering & data ) [inline]`
- 0.14.27.3.29** `operator<<()` [10/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const IndexedRootRelation & data ) [inline]`
- 0.14.27.3.30** `operator<<()` [11/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const PolyRef & data ) [inline]`
- 0.14.27.3.31** `operator<<()` [12/14] `std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const RootFunction & data ) [inline]`

**0.14.27.3.32 operator<<()** [13/14] std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const SymbolicInterval & data ) [inline]

**0.14.27.3.33 operator<<()** [14/14] std::ostream& smtrat::cadcells::datastructures::operator<< ( std::ostream & os, const TaggedIndexedRoot & data ) [inline]

**0.14.27.3.34 operator==()** [1/8] bool smtrat::cadcells::datastructures::operator== ( const CompoundMax & lhs, const CompoundMax & rhs ) [inline]

**0.14.27.3.35 operator==()** [2/8] bool smtrat::cadcells::datastructures::operator== ( const CompoundMin & lhs, const CompoundMin & rhs ) [inline]

**0.14.27.3.36 operator==()** [3/8] template<typename P> bool smtrat::cadcells::datastructures::operator== ( const DerivationRef< P > & lhs, const DerivationRef< P > & rhs )

**0.14.27.3.37 operator==()** [4/8] bool smtrat::cadcells::datastructures::operator== ( const IndexedRoot & lhs, const IndexedRoot & rhs ) [inline]

**0.14.27.3.38 operator==()** [5/8] bool smtrat::cadcells::datastructures::operator== ( const IndexedRootRelation & lhs, const IndexedRootRelation & rhs ) [inline]

**0.14.27.3.39 operator==()** [6/8] bool smtrat::cadcells::datastructures::operator== ( const PolyRef & lhs, const PolyRef & rhs ) [inline]

**0.14.27.3.40 operator==()** [7/8] bool smtrat::cadcells::datastructures::operator== ( const RootFunction & lhs, const RootFunction & rhs ) [inline]

**0.14.27.3.41 operator==()** [8/8] bool smtrat::cadcells::datastructures::operator== ( const TaggedIndexedRoot & lhs, const TaggedIndexedRoot & rhs ) [inline]

**0.14.27.3.42 prop\_get()** [1/2] template<class T, class... Ts> const auto& smtrat::cadcells::datastructures::prop\_get ( const Propertiest< T, Ts... > & sets )

**0.14.27.3.43 prop\_get() [2/2]** template<class S , class T , class... Ts, typename std::enable\_if<!std::is\_same< S, T >::value>::type>  
const auto& smtrat::cadcells::datastructures::prop\_get (  
 const PropertiesT< T, Ts... > & sets )

**0.14.27.3.44 prop\_has() [1/2]** template<class S , class T , class... Ts, typename std::enable\_if<!std::is\_same< S, T >::value>::type>  
bool smtrat::cadcells::datastructures::prop\_has (  
 const PropertiesT< T, Ts... > & sets,  
 const S & element )

**0.14.27.3.45 prop\_has() [2/2]** template<class T , class... Ts>  
bool smtrat::cadcells::datastructures::prop\_has (  
 const PropertiesT< T, Ts... > & sets,  
 const T & element )

**0.14.27.3.46 prop\_insert() [1/2]** template<class S , class T , class... Ts, typename std::enable\_if<!std::is\_same< S, T >::value>::type>  
void smtrat::cadcells::datastructures::prop\_insert (  
 PropertiesT< T, Ts... > & sets,  
 const S & element )

**0.14.27.3.47 prop\_insert() [2/2]** template<class T , class... Ts>  
void smtrat::cadcells::datastructures::prop\_insert (  
 PropertiesT< T, Ts... > & sets,  
 const T & element )

**0.14.27.3.48 upper\_lt\_lower()** bool smtrat::cadcells::datastructures::upper\_lt\_lower (  
 const DelineationInterval & del1,  
 const DelineationInterval & del2 ) [inline]

Compares an upper bound with a lower bound of DelineationIntervals.

It respects whether the interval is a section or sector.

**0.14.27.3.49 upper\_lt\_upper()** bool smtrat::cadcells::datastructures::upper\_lt\_upper (  
 const DelineationInterval & del1,  
 const DelineationInterval & del2 ) [inline]

Compares the upper bounds of two DelineationIntervals.

It respects whether the interval is a section or sector.

## 0.14.28 smtrat::cadcells::datastructures::detail Namespace Reference

### Data Structures

- struct [PolyProperties](#)
- struct [AssignmentProperties](#)

## 0.14.29 smtrat::cadcells::helper Namespace Reference

### Functions

- [MultivariateRoot as\\_multivariate\\_root](#) (const [datastructures::PolyPool](#) &pool, carl::Variable main\_var, [datastructures::IndexedRoot](#) r)

- std::vector< Atom > **to\_formula** (const **datastructures::PolyPool** &pool, carl::Variable main\_var, const **datastructures::SymbolicInterval** &c)
 

*Converts a **datastructures::SymbolicInterval** to an **Atom**.*

### 0.14.29.1 Function Documentation

**0.14.29.1.1 as\_multivariate\_root()** **MultivariateRoot** smtrat::cadcells::helper::as\_multivariate\_root (

```
 const datastructures::PolyPool & pool,
 carl::Variable main_var,
 datastructures::IndexedRoot r) [inline]
```

Converts an **datastructures::IndexedRoot** to a **MultivariateRoot**.

**0.14.29.1.2 to\_formula()** std::vector<**Atom**> smtrat::cadcells::helper::to\_formula (

```
 const datastructures::PolyPool & pool,
 carl::Variable main_var,
 const datastructures::SymbolicInterval & c)
```

Converts a **datastructures::SymbolicInterval** to an **Atom**.

## 0.14.30 smtrat::cadcells::operators Namespace Reference

Projection operators.

### Namespaces

- **mccallum\_filtered\_impl**
- **properties**

*Contains all properties that are stored in a derivation.*
- **rules**

*Implementation of derivation rules.*

### Data Structures

- struct **PropertiesSet**
- struct **PropertiesSet< op::mccallum >**

### Enumerations

- enum **op** {
 mccallum , mccallum\_filtered , mccallum\_filtered\_all , mccallum\_filtered\_bounds ,
 mccallum\_filtered\_samples , mccallum\_filtered\_all\_selective }

### Functions

- std::ostream & **operator<<** (std::ostream &os, **op op**)
- template<op Op>
 

```
bool project_basic_properties (datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > &deriv)
```

*Project basic properties, i.e.*
- template<op Op>
 

```
void delineate_properties (datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > &deriv)
```

*Delineate properties.*

- template<op Op>  
void **delineate\_properties** (datastructures::SampledDerivation< typename PropertiesSet< Op >::type > &deriv)  
*Delineate properties.*
- template<op Op>  
bool **project\_delineated\_cell\_properties** (datastructures::CellRepresentation< typename PropertiesSet< Op >::type > &deriv, bool cell\_represents=true)  
*Project cell properties that depend on a delineation.*
- template<op Op>  
bool **project\_cell\_properties** (datastructures::SampledDerivation< typename PropertiesSet< Op >::type > &deriv)  
*Project cell properties.*
- template<op Op>  
bool **project\_covering\_properties** (datastructures::CoveringRepresentation< typename PropertiesSet< Op >::type > &repr)  
*Project covering properties.*
- template<> bool **project\_basic\_properties< op::mccallum >** (datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &deriv)
- template<> void **delineate\_properties< op::mccallum >** (datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &deriv)
- template<> void **delineate\_properties< op::mccallum >** (datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > &deriv)
- template<> bool **project\_delineated\_cell\_properties< op::mccallum >** (datastructures::CellRepresentation< PropertiesSet< op::mccallum >::type > &repr, bool cell\_represents)
- template<> bool **project\_cell\_properties< op::mccallum >** (datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > &deriv)
- template<> bool **project\_covering\_properties< op::mccallum >** (datastructures::CoveringRepresentation< PropertiesSet< op::mccallum >::type > &repr)
- template<> void **delineate\_properties< op::mccallum\_filtered >** (datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered >::type > &deriv)
- template<> void **delineate\_properties< op::mccallum\_filtered\_all >** (datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered\_all >::type > &deriv)
- template<> void **delineate\_properties< op::mccallum\_filtered\_bounds >** (datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered\_bounds >::type > &deriv)
- template<> void **delineate\_properties< op::mccallum\_filtered\_samples >** (datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered\_samples >::type > &deriv)
- template<> void **delineate\_properties< op::mccallum\_filtered\_all\_selective >** (datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered\_all\_selective >::type > &deriv)

## Variables

- static const char \* OpStrings [] = { "mccallum", "mccallum\_filtered", "mccallum\_filtered\_all", "mccallum\_filtered\_bounds", "mccallum\_filtered\_samples", "mccallum\_filtered\_all\_selective" }

### 0.14.30.1 Detailed Description

Projection operators.

Currently, only the McCallum based operator is implemented.

#### 0.14.30.1.1 Relevant projection functions

We refer to [algorithms](#) for usage examples.

##### 0.14.30.1.1.1 Cells

- project\_cell\_properties
- project\_basic\_properties
- delineate\_properties
- project\_delineated\_cell\_properties

### 0.14.30.1.1.2 Coverings

- project\_cell\_properties (on each cell individually)
- project\_basic\_properties (on each cell individually)
- delineate\_properties (on each cell individually)
- project\_covering\_properties

### 0.14.30.1.1.3 Delineation

- project\_basic\_properties
- delineate\_properties
- project\_delineation\_properties

## 0.14.30.2 Enumeration Type Documentation

### 0.14.30.2.1 op enum smtrat::cadcells::operators::op

Enumerator

|                                 |  |
|---------------------------------|--|
| mccallum                        |  |
| mccallum_filtered               |  |
| mccallum_filtered_all           |  |
| mccallum_filtered_bounds        |  |
| mccallum_filtered_samples       |  |
| mccallum_filtered_all_selective |  |

## 0.14.30.3 Function Documentation

### 0.14.30.3.1 delineate\_properties() [1/2] template<op Op>

```
void smtrat::cadcells::operators::delineate_properties (
 datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > &
deriv) [inline]
Delineate properties.
```

### 0.14.30.3.2 delineate\_properties() [2/2] template<op Op>

```
void smtrat::cadcells::operators::delineate_properties (
 datastructures::SampledDerivation< typename PropertiesSet< Op >::type > & deriv
) [inline]
Delineate properties.
```

### 0.14.30.3.3 delineate\_properties< op::mccallum >() [1/2] template<>

```
void smtrat::cadcells::operators::delineate_properties< op::mccallum > (
 datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > &
deriv) [inline]
```

**0.14.30.3.4 delineate\_properties< op::mccallum >()** [2/2] template<>  
void smtrat::cadcells::operators::delineate\_properties< op::mccallum > ( datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > & deriv ) [inline]

**0.14.30.3.5 delineate\_properties< op::mccallum\_filtered >()** template<>  
void smtrat::cadcells::operators::delineate\_properties< op::mccallum\_filtered > ( datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered >::type > & deriv ) [inline]

**0.14.30.3.6 delineate\_properties< op::mccallum\_filtered\_all >()** template<>  
void smtrat::cadcells::operators::delineate\_properties< op::mccallum\_filtered\_all > ( datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered\_all >::type > & deriv ) [inline]

**0.14.30.3.7 delineate\_properties< op::mccallum\_filtered\_all\_selective >()** template<>  
void smtrat::cadcells::operators::delineate\_properties< op::mccallum\_filtered\_all\_selective > ( datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered\_all\_selective >::type > & deriv ) [inline]

**0.14.30.3.8 delineate\_properties< op::mccallum\_filtered\_bounds >()** template<>  
void smtrat::cadcells::operators::delineate\_properties< op::mccallum\_filtered\_bounds > ( datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered\_bounds >::type > & deriv ) [inline]

**0.14.30.3.9 delineate\_properties< op::mccallum\_filtered\_samples >()** template<>  
void smtrat::cadcells::operators::delineate\_properties< op::mccallum\_filtered\_samples > ( datastructures::SampledDerivation< PropertiesSet< op::mccallum\_filtered\_samples >::type > & deriv ) [inline]

**0.14.30.3.10 operator<<()** std::ostream& smtrat::cadcells::operators::operator<< ( std::ostream & os, op op ) [inline]

**0.14.30.3.11 project\_basic\_properties()** template<op Op>  
bool smtrat::cadcells::operators::project\_basic\_properties ( datastructures::DelineatedDerivation< typename PropertiesSet< Op >::type > & deriv ) [inline]

Project basic properties, i.e.  
independent from any sample or delineation.

**0.14.30.3.12 project\_basic\_properties< op::mccallum >()** template<>  
bool smtrat::cadcells::operators::project\_basic\_properties< op::mccallum > ( datastructures::DelineatedDerivation< PropertiesSet< op::mccallum >::type > & deriv ) [inline]

**0.14.30.3.13 `project_cell_properties()`** template<op Op>  
`bool smtrat::cadcells::operators::project_cell_properties (`  
 `datastructures::SampledDerivation< typename PropertiesSet< Op >::type > & deriv`  
`) [inline]`  
 Project cell properties.  
 Returns false iff the given operator is incomplete and cannot cover the given case (i.e. on nullification with McCallum).

**0.14.30.3.14 `project_cell_properties< op::mccallum >()`** template<>  
`bool smtrat::cadcells::operators::project_cell_properties< op::mccallum > (`  
 `datastructures::SampledDerivation< PropertiesSet< op::mccallum >::type > & deriv`  
`) [inline]`

**0.14.30.3.15 `project_covering_properties()`** template<op Op>  
`bool smtrat::cadcells::operators::project_covering_properties (`  
 `datastructures::CoveringRepresentation< typename PropertiesSet< Op >::type > &`  
`repr ) [inline]`  
 Project covering properties.

**0.14.30.3.16 `project_covering_properties< op::mccallum >()`** template<>  
`bool smtrat::cadcells::operators::project_covering_properties< op::mccallum > (`  
 `datastructures::CoveringRepresentation< PropertiesSet< op::mccallum >::type > &`  
`repr ) [inline]`

**0.14.30.3.17 `project_delineated_cell_properties()`** template<op Op>  
`bool smtrat::cadcells::operators::project_delineated_cell_properties (`  
 `datastructures::CellRepresentation< typename PropertiesSet< Op >::type > & deriv,`  
 `bool cell_represents = true ) [inline]`  
 Project cell properties that depend on a delineation.

**0.14.30.3.18 `project_delineated_cell_properties< op::mccallum >()`** template<>  
`bool smtrat::cadcells::operators::project_delineated_cell_properties< op::mccallum > (`  
 `datastructures::CellRepresentation< PropertiesSet< op::mccallum >::type > &`  
`repr,`  
 `bool cell_represents ) [inline]`

## 0.14.30.4 Variable Documentation

**0.14.30.4.1 `OpStrings`** const char\* smtrat::cadcells::operators::OpStrings[ ] = { "mccallum",  
`"mccallum_filtered", "mccallum_filtered_all", "mccallum_filtered_bounds", "mccallum_filtered_samples",`  
`"mccallum_filtered_all_selective" } [static]`

## 0.14.31 `smtrat::cadcells::operators::mccallum_filtered_impl` Namespace Reference

### Data Structures

- struct `PropertiesSet`

### Functions

- bool `project_basic_properties (datastructures::DelineatedDerivation< PropertiesSet::type > &deriv)`

- void [delineate\\_properties](#) ([datastructures::DelineatedDerivation< PropertiesSet::type >](#) &deriv)
- bool [project\\_delineated\\_cell\\_properties](#) ([datastructures::CellRepresentation< PropertiesSet::type >](#) &repr,  
bool cell\_represents)
- bool [project\\_cell\\_properties](#) ([datastructures::SampledDerivation< PropertiesSet::type >](#) &deriv)
- bool [project\\_covering\\_properties](#) ([datastructures::CoveringRepresentation< PropertiesSet::type >](#) &repr)

#### 0.14.31.1 Function Documentation

**0.14.31.1.1 [delineate\\_properties\(\)](#)** void smtrat::cadcells::operators::mccallum\_filtered\_impl<  
::delineate\_properties (

[datastructures::DelineatedDerivation< PropertiesSet::type >](#) & deriv ) [inline]

**0.14.31.1.2 [project\\_basic\\_properties\(\)](#)** bool smtrat::cadcells::operators::mccallum\_filtered\_impl<  
::project\_basic\_properties (

[datastructures::DelineatedDerivation< PropertiesSet::type >](#) & deriv ) [inline]

**0.14.31.1.3 [project\\_cell\\_properties\(\)](#)** bool smtrat::cadcells::operators::mccallum\_filtered\_impl<  
::project\_cell\_properties (

[datastructures::SampledDerivation< PropertiesSet::type >](#) & deriv ) [inline]

**0.14.31.1.4 [project\\_covering\\_properties\(\)](#)** bool smtrat::cadcells::operators::mccallum\_filtered\_<  
impl::project\_covering\_properties (

[datastructures::CoveringRepresentation< PropertiesSet::type >](#) & repr ) [inline]

**0.14.31.1.5 [project\\_delineated\\_cell\\_properties\(\)](#)** bool smtrat::cadcells::operators::mccallum\_<  
filtered\_impl::project\_delineated\_cell\_properties (

[datastructures::CellRepresentation< PropertiesSet::type >](#) & repr,

bool cell\_represents ) [inline]

### 0.14.32 smtrat::cadcells::operators::properties Namespace Reference

Contains all properties that are stored in a derivation.

#### Data Structures

- struct [poly\\_sgn\\_inv](#)
- struct [poly\\_irreducible\\_sgn\\_inv](#)
- struct [poly\\_semi\\_sgn\\_inv](#)
- struct [poly\\_irreducible\\_semi\\_sgn\\_inv](#)
- struct [poly\\_ord\\_inv](#)
- struct [root\\_well\\_def](#)
- struct [poly\\_pdel](#)
- struct [cell\\_connected](#)
- struct [poly\\_additional\\_root\\_outside](#)
- struct [poly\\_ord\\_inv\\_base](#)
- struct [root\\_ordering\\_holds](#)

## Functions

- bool `operator==` (const `poly_sgn_inv` &lhs, const `poly_sgn_inv` &rhs)
- bool `operator<` (const `poly_sgn_inv` &lhs, const `poly_sgn_inv` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `poly_sgn_inv` &data)
- bool `operator==` (const `poly_irreducible_sgn_inv` &lhs, const `poly_irreducible_sgn_inv` &rhs)
- bool `operator<` (const `poly_irreducible_sgn_inv` &lhs, const `poly_irreducible_sgn_inv` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `poly_irreducible_sgn_inv` &data)
- bool `operator==` (const `poly_semi_sgn_inv` &lhs, const `poly_semi_sgn_inv` &rhs)
- bool `operator<` (const `poly_semi_sgn_inv` &lhs, const `poly_semi_sgn_inv` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `poly_semi_sgn_inv` &data)
- bool `operator==` (const `poly_irreducible_semi_sgn_inv` &lhs, const `poly_irreducible_semi_sgn_inv` &rhs)
- bool `operator<` (const `poly_irreducible_semi_sgn_inv` &lhs, const `poly_irreducible_semi_sgn_inv` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `poly_irreducible_semi_sgn_inv` &data)
- bool `operator==` (const `poly_ord_inv` &lhs, const `poly_ord_inv` &rhs)
- bool `operator<` (const `poly_ord_inv` &lhs, const `poly_ord_inv` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `poly_ord_inv` &data)
- bool `operator==` (const `root_well_def` &lhs, const `root_well_def` &rhs)
- bool `operator<` (const `root_well_def` &lhs, const `root_well_def` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `root_well_def` &data)
- bool `operator==` (const `poly_pdel` &lhs, const `poly_pdel` &rhs)
- bool `operator<` (const `poly_pdel` &lhs, const `poly_pdel` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `poly_pdel` &data)
- bool `operator==` (const `cell_connected` &lhs, const `cell_connected` &rhs)
- bool `operator<` (const `cell_connected` &lhs, const `cell_connected` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `cell_connected` &data)
- bool `operator==` (const `poly_additional_root_outside` &lhs, const `poly_additional_root_outside` &rhs)
- bool `operator<` (const `poly_additional_root_outside` &lhs, const `poly_additional_root_outside` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `poly_additional_root_outside` &data)
- bool `operator==` (const `poly_ord_inv_base` &lhs, const `poly_ord_inv_base` &rhs)
- bool `operator<` (const `poly_ord_inv_base` &lhs, const `poly_ord_inv_base` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `poly_ord_inv_base` &data)
- bool `operator==` (const `root_ordering_holds` &lhs, const `root_ordering_holds` &rhs)
- bool `operator<` (const `root_ordering_holds` &lhs, const `root_ordering_holds` &rhs)
- std::ostream & `operator<<` (std::ostream &os, const `root_ordering_holds` &data)

### 0.14.32.1 Detailed Description

Contains all properties that are stored in a derivation.

Note that not all properties have a representation here as not all of them are stored but resolved directly in the derivation rules.

### 0.14.32.2 Function Documentation

```
0.14.32.2.1 operator<() [1/11] bool smrat::cadcells::operators::properties::operator< (
 const cell_connected & lhs,
 const cell_connected & rhs) [inline]
```

```
0.14.32.2.2 operator<() [2/11] bool smrat::cadcells::operators::properties::operator< (
 const poly_additional_root_outside & lhs,
 const poly_additional_root_outside & rhs) [inline]
```

```
0.14.32.2.3 operator<() [3/11] bool smtrat::cadcells::operators::properties::operator< (
 const poly_irreducible_semi_sgn_inv & lhs,
 const poly_irreducible_semi_sgn_inv & rhs) [inline]

0.14.32.2.4 operator<() [4/11] bool smtrat::cadcells::operators::properties::operator< (
 const poly_irreducible_sgn_inv & lhs,
 const poly_irreducible_sgn_inv & rhs) [inline]

0.14.32.2.5 operator<() [5/11] bool smtrat::cadcells::operators::properties::operator< (
 const poly_ord_inv & lhs,
 const poly_ord_inv & rhs) [inline]

0.14.32.2.6 operator<() [6/11] bool smtrat::cadcells::operators::properties::operator< (
 const poly_ord_inv_base & lhs,
 const poly_ord_inv_base & rhs) [inline]

0.14.32.2.7 operator<() [7/11] bool smtrat::cadcells::operators::properties::operator< (
 const poly_pdel & lhs,
 const poly_pdel & rhs) [inline]

0.14.32.2.8 operator<() [8/11] bool smtrat::cadcells::operators::properties::operator< (
 const poly_semi_sgn_inv & lhs,
 const poly_semi_sgn_inv & rhs) [inline]

0.14.32.2.9 operator<() [9/11] bool smtrat::cadcells::operators::properties::operator< (
 const poly_sgn_inv & lhs,
 const poly_sgn_inv & rhs) [inline]

0.14.32.2.10 operator<() [10/11] bool smtrat::cadcells::operators::properties::operator< (
 const root_ordering_holds & lhs,
 const root_ordering_holds & rhs) [inline]

0.14.32.2.11 operator<() [11/11] bool smtrat::cadcells::operators::properties::operator< (
 const root_well_def & lhs,
 const root_well_def & rhs) [inline]

0.14.32.2.12 operator<<() [1/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const cell_connected & data) [inline]

0.14.32.2.13 operator<<() [2/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const poly_additional_root_outside & data) [inline]
```

```
0.14.32.2.14 operator<<() [3/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const poly_irreducible_semi_sgn_inv & data) [inline]
```

  

```
0.14.32.2.15 operator<<() [4/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const poly_irreducible_sgn_inv & data) [inline]
```

  

```
0.14.32.2.16 operator<<() [5/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const poly_ord_inv & data) [inline]
```

  

```
0.14.32.2.17 operator<<() [6/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const poly_ord_inv_base & data) [inline]
```

  

```
0.14.32.2.18 operator<<() [7/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const poly_pdel & data) [inline]
```

  

```
0.14.32.2.19 operator<<() [8/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const poly_semi_sgn_inv & data) [inline]
```

  

```
0.14.32.2.20 operator<<() [9/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const poly_sgn_inv & data) [inline]
```

  

```
0.14.32.2.21 operator<<() [10/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const root_ordering_holds & data) [inline]
```

  

```
0.14.32.2.22 operator<<() [11/11] std::ostream& smtrat::cadcells::operators::properties::operator<<
(
 std::ostream & os,
 const root_well_def & data) [inline]
```

- 0.14.32.2.23** `operator==() [1/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const cell_connected & lhs,`  
`const cell_connected & rhs ) [inline]`
- 0.14.32.2.24** `operator==() [2/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const poly_additional_root_outside & lhs,`  
`const poly_additional_root_outside & rhs ) [inline]`
- 0.14.32.2.25** `operator==() [3/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const poly_irreducible_semi_sgn_inv & lhs,`  
`const poly_irreducible_semi_sgn_inv & rhs ) [inline]`
- 0.14.32.2.26** `operator==() [4/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const poly_irreducible_sgn_inv & lhs,`  
`const poly_irreducible_sgn_inv & rhs ) [inline]`
- 0.14.32.2.27** `operator==() [5/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const poly_ord_inv & lhs,`  
`const poly_ord_inv & rhs ) [inline]`
- 0.14.32.2.28** `operator==() [6/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const poly_ord_inv_base & lhs,`  
`const poly_ord_inv_base & rhs ) [inline]`
- 0.14.32.2.29** `operator==() [7/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const poly_pdel & lhs,`  
`const poly_pdel & rhs ) [inline]`
- 0.14.32.2.30** `operator==() [8/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const poly_semi_sgn_inv & lhs,`  
`const poly_semi_sgn_inv & rhs ) [inline]`
- 0.14.32.2.31** `operator==() [9/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const poly_sgn_inv & lhs,`  
`const poly_sgn_inv & rhs ) [inline]`
- 0.14.32.2.32** `operator==() [10/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const root_ordering_holds & lhs,`  
`const root_ordering_holds & rhs ) [inline]`
- 0.14.32.2.33** `operator==() [11/11]` `bool smtrat::cadcells::operators::properties::operator== (`  
`const root_well_def & lhs,`  
`const root_well_def & rhs ) [inline]`

### 0.14.33 smrat::cadcells::operators::rules Namespace Reference

Implementation of derivation rules.

#### Namespaces

- [additional\\_root\\_outside\\_util](#)
- [filter\\_util](#)
- [ordering\\_util](#)

#### Functions

- template<typename P >  
void [root\\_well\\_def](#) ([datastructures::SampledDerivation](#)< P > &deriv, [datastructures::IndexedRoot](#) root)
- template<typename P >  
bool [is\\_trivial\\_root\\_well\\_def](#) ([datastructures::SampledDerivation](#)< P > &deriv, [datastructures::IndexedRoot](#) root)
- template<typename P >  
bool [poly\\_non\\_null](#) ([datastructures::SampledDerivation](#)< P > &deriv, [datastructures::PolyRef](#) poly)
- template<typename P >  
bool [poly\\_pdel](#) ([datastructures::SampledDerivation](#)< P > &deriv, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [poly\\_irreducible\\_ord\\_inv](#) ([datastructures::SampledDerivation](#)< P > &deriv, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [poly\\_ord\\_inv](#) ([datastructures::SampledDerivation](#)< P > &deriv, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [poly\\_ord\\_inv\\_base](#) ([datastructures::SampledDerivation](#)< P > &deriv, const [datastructures::SymbolicInterval](#) &cell, const [datastructures::IndexedRootOrdering](#) &, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [poly\\_sgn\\_inv](#) ([datastructures::DelineatedDerivation](#)< P > &deriv, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [poly\\_semi\\_sgn\\_inv](#) ([datastructures::DelineatedDerivation](#)< P > &deriv, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [poly\\_irreducible\\_nonzero\\_sgn\\_inv](#) ([datastructures::DelineatedDerivation](#)< P > &deriv, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [poly\\_irreducible\\_nonzero\\_semi\\_sgn\\_inv](#) ([datastructures::DelineatedDerivation](#)< P > &deriv, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [delineate](#) ([datastructures::DelineatedDerivation](#)< P > &deriv, const [properties::poly\\_irreducible\\_sgn\\_inv](#) &prop)
- template<typename P >  
void [delineate](#) ([datastructures::DelineatedDerivation](#)< P > &deriv, const [properties::poly\\_irreducible\\_semi\\_sgn\\_inv](#) &prop)
- template<typename P >  
void [cell\\_connected](#) ([datastructures::SampledDerivation](#)< P > &deriv, const [datastructures::SymbolicInterval](#) &cell, const [datastructures::IndexedRootOrdering](#) &ordering)
- template<typename P >  
void [cell\\_analytic\\_submanifold](#) ([[maybe\_unused]] [datastructures::SampledDerivation](#)< P > &deriv, const [datastructures::SymbolicInterval](#) &)
- template<typename P >  
void [poly\\_irreducible\\_sgn\\_inv\\_ec](#) ([datastructures::SampledDerivation](#)< P > &deriv, const [datastructures::SymbolicInterval](#) &cell, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [poly\\_irreducible\\_semi\\_sgn\\_inv\\_ec](#) ([datastructures::SampledDerivation](#)< P > &deriv, const [datastructures::SymbolicInterval](#) &cell, [datastructures::PolyRef](#) poly)
- template<typename P >  
void [root\\_represents](#) ([datastructures::SampledDerivation](#)< P > &deriv, [datastructures::IndexedRoot](#) root)

- template<typename P >  
void `cell_represents`(`datastructures::SampledDerivation`< P > &deriv, const `datastructures::SymbolicInterval` &cell)
- template<typename P >  
void `root_ordering_holds`(`datastructures::SampledDerivation`< P > &deriv, const `datastructures::IndexedRootOrdering` &ordering)
- template<typename P >  
void `poly_additional_root_outside`(`datastructures::SampledDerivation`< P > &deriv, const `datastructures::SymbolicInterval` &cell, const `datastructures::IndexedRootOrdering` &ordering, `datastructures::PolyRef` poly)
- template<typename P >  
void `poly_irreducible_sgn_inv`(`datastructures::SampledDerivation`< P > &deriv, const `datastructures::SymbolicInterval` &cell, const `datastructures::IndexedRootOrdering` &ordering, `datastructures::PolyRef` poly)
- template<typename P >  
void `poly_irreducible_semi_sgn_inv`(`datastructures::SampledDerivation`< P > &deriv, const `datastructures::SymbolicInterval` &cell, const `datastructures::IndexedRootOrdering` &ordering, `datastructures::PolyRef` poly)
- template<typename P >  
void `poly_irreducible_sgn_inv_filtered`(`datastructures::SampledDerivation`< P > &deriv, const `datastructures::SymbolicInterval` &, const `datastructures::IndexedRootOrdering` &, `datastructures::PolyRef` poly)
- template<typename P >  
void `poly_irreducible_semi_sgn_inv_filtered`(`datastructures::SampledDerivation`< P > &deriv, const `datastructures::SymbolicInterval` &, const `datastructures::IndexedRootOrdering` &, `datastructures::PolyRef` poly)
- template<typename P >  
void `covering_holds`(`datastructures::DelineatedDerivation`< P > &, const `datastructures::CoveringDescription` &covering, const `datastructures::IndexedRootOrdering` &ordering)
- template<typename P >  
void `delineate_all_compound`(`datastructures::SampledDerivation`< P > &deriv, const `properties::root_ordering_holds` &prop)
- template<typename P >  
void `delineate_all`(`datastructures::SampledDerivation`< P > &deriv, const `properties::root_ordering_holds` &prop)
- template<typename P >  
void `delineate_samples`(`datastructures::SampledDerivation`< P > &deriv, const `properties::root_ordering_holds` &prop)
- template<typename P >  
void `delineate_all_selective`(`datastructures::SampledDerivation`< P > &deriv, const `properties::root_ordering_holds` &prop)
- template<typename P >  
void `delineate_bounds`(`datastructures::SampledDerivation`< P > &deriv, const `properties::root_ordering_holds` &prop)
- template<typename P >  
void `delineate_noop`(`datastructures::SampledDerivation`< P > &deriv, const `properties::root_ordering_holds` &prop)
- template<typename P >  
bool `root_ordering_holds_delineated`(`datastructures::SampledDerivation`< P > &deriv, const `datastructures::SymbolicInterval` &, const `datastructures::IndexedRootOrdering` &underlying\_ordering, const `datastructures::IndexedRootOrdering` &ordering)

#### 0.14.33.1 Detailed Description

Implementation of derivation rules.

Each rule is realized by a function which works on a derivation object. The parameters of the respective properties are passed as function parameter. The function realizing a derivation rule might either call other derivation rules or add properties to the given derivation.

#### 0.14.33.2 Function Documentation

---

**0.14.33.2.1 `cell_analytic_submanifold()`** template<typename P >  
void smtrat::cadcells::operators::rules::cell\_analytic\_submanifold ( [maybe\_unused] ] `datastructures::SampledDerivation< P >` & deriv,  
const `datastructures::SymbolicInterval` & )

**0.14.33.2.2 `cell_connected()`** template<typename P >  
void smtrat::cadcells::operators::rules::cell\_connected ( `datastructures::SampledDerivation< P >` & deriv,  
const `datastructures::SymbolicInterval` & cell,  
const `datastructures::IndexedRootOrdering` & ordering )

**0.14.33.2.3 `cell_represents()`** template<typename P >  
void smtrat::cadcells::operators::rules::cell\_represents ( `datastructures::SampledDerivation< P >` & deriv,  
const `datastructures::SymbolicInterval` & cell )

**0.14.33.2.4 `covering_holds()`** template<typename P >  
void smtrat::cadcells::operators::rules::covering\_holds ( `datastructures::DelineatedDerivation< P >` & ,  
const `datastructures::CoveringDescription` & covering,  
const `datastructures::IndexedRootOrdering` & ordering )

**0.14.33.2.5 `delineate()` [1/2]** template<typename P >  
void smtrat::cadcells::operators::rules::delineate ( `datastructures::DelineatedDerivation< P >` & deriv,  
const `properties::poly_irreducible_semi_sgn_inv` & prop )

**0.14.33.2.6 `delineate()` [2/2]** template<typename P >  
void smtrat::cadcells::operators::rules::delineate ( `datastructures::DelineatedDerivation< P >` & deriv,  
const `properties::poly_irreducible_sgn_inv` & prop )

**0.14.33.2.7 `delineate_all()`** template<typename P >  
void smtrat::cadcells::operators::rules::delineate\_all ( `datastructures::SampledDerivation< P >` & deriv,  
const `properties::root_ordering_holds` & prop )

**0.14.33.2.8 `delineate_all_compound()`** template<typename P >  
void smtrat::cadcells::operators::rules::delineate\_all\_compound ( `datastructures::SampledDerivation< P >` & deriv,  
const `properties::root_ordering_holds` & prop )

**0.14.33.2.9 `delineate_all_selective()`** template<typename P >  
void smtrat::cadcells::operators::rules::delineate\_all\_selective ( `datastructures::SampledDerivation< P >` & deriv,  
const `properties::root_ordering_holds` & prop )

**0.14.33.2.10 delineate\_bounds()** template<typename P >  
void smtrat::cadcells::operators::rules::delineate\_bounds (  
 datastructures::SampledDerivation< P > & deriv,  
 const properties::root\_ordering\_holds & prop )

**0.14.33.2.11 delineate\_noop()** template<typename P >  
void smtrat::cadcells::operators::rules::delineate\_noop (  
 datastructures::SampledDerivation< P > & deriv,  
 const properties::root\_ordering\_holds & prop )

**0.14.33.2.12 delineate\_samples()** template<typename P >  
void smtrat::cadcells::operators::rules::delineate\_samples (  
 datastructures::SampledDerivation< P > & deriv,  
 const properties::root\_ordering\_holds & prop )

**0.14.33.2.13 is\_trivial\_root\_well\_def()** template<typename P >  
bool smtrat::cadcells::operators::is\_trivial\_root\_well\_def (  
 datastructures::SampledDerivation< P > & deriv,  
 datastructures::IndexedRoot root )

**0.14.33.2.14 poly\_additional\_root\_outside()** template<typename P >  
void smtrat::cadcells::operators::poly\_additional\_root\_outside (  
 datastructures::SampledDerivation< P > & deriv,  
 const datastructures::SymbolicInterval & cell,  
 const datastructures::IndexedRootOrdering & ordering,  
 datastructures::PolyRef poly )

**0.14.33.2.15 poly\_irreducible\_nonzero\_semi\_sgn\_inv()** template<typename P >  
void smtrat::cadcells::operators::poly\_irreducible\_nonzero\_semi\_sgn\_inv (  
 datastructures::DelineatedDerivation< P > & deriv,  
 datastructures::PolyRef poly )

**0.14.33.2.16 poly\_irreducible\_nonzero\_sgn\_inv()** template<typename P >  
void smtrat::cadcells::operators::poly\_irreducible\_nonzero\_sgn\_inv (  
 datastructures::DelineatedDerivation< P > & deriv,  
 datastructures::PolyRef poly )

**0.14.33.2.17 poly\_irreducible\_ord\_inv()** template<typename P >  
void smtrat::cadcells::operators::poly\_irreducible\_ord\_inv (  
 datastructures::SampledDerivation< P > & deriv,  
 datastructures::PolyRef poly )

**0.14.33.2.18 poly\_irreducible\_semi\_sgn\_inv()** template<typename P >  
void smtrat::cadcells::operators::poly\_irreducible\_semi\_sgn\_inv (  
 datastructures::SampledDerivation< P > & deriv,  
 const datastructures::SymbolicInterval & cell,  
 const datastructures::IndexedRootOrdering & ordering,  
 datastructures::PolyRef poly )

```
0.14.33.2.19 poly_irreducible_semi_sgn_inv_ec() template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_semi_sgn_inv_ec (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::SymbolicInterval & cell,
 datastructures::PolyRef poly)
```

```
0.14.33.2.20 poly_irreducible_semi_sgn_inv_filtered() template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_semi_sgn_inv_filtered (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::SymbolicInterval & ,
 const datastructures::IndexedRootOrdering & ,
 datastructures::PolyRef poly)
```

```
0.14.33.2.21 poly_irreducible_sgn_inv() template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::SymbolicInterval & cell,
 const datastructures::IndexedRootOrdering & ordering,
 datastructures::PolyRef poly)
```

```
0.14.33.2.22 poly_irreducible_sgn_inv_ec() template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv_ec (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::SymbolicInterval & cell,
 datastructures::PolyRef poly)
```

```
0.14.33.2.23 poly_irreducible_sgn_inv_filtered() template<typename P >
void smtrat::cadcells::operators::rules::poly_irreducible_sgn_inv_filtered (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::SymbolicInterval & ,
 const datastructures::IndexedRootOrdering & ,
 datastructures::PolyRef poly)
```

```
0.14.33.2.24 poly_non_null() template<typename P >
bool smtrat::cadcells::operators::rules::poly_non_null (
 datastructures::SampledDerivation< P > & deriv,
 datastructures::PolyRef poly)
```

```
0.14.33.2.25 poly_ord_inv() template<typename P >
void smtrat::cadcells::operators::rules::poly_ord_inv (
 datastructures::SampledDerivation< P > & deriv,
 datastructures::PolyRef poly)
```

```
0.14.33.2.26 poly_ord_inv_base() template<typename P >
void smtrat::cadcells::operators::rules::poly_ord_inv_base (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::SymbolicInterval & cell,
 const datastructures::IndexedRootOrdering & ,
 datastructures::PolyRef poly)
```

```
0.14.33.2.27 poly_pdel() template<typename P >
bool smtrat::cadcells::operators::rules::poly_pdel (
 datastructures::SampledDerivation< P > & deriv,
 datastructures::PolyRef poly)
```

```
0.14.33.2.28 poly_semi_sgn_inv() template<typename P >
void smtrat::cadcells::operators::rules::poly_semi_sgn_inv (
 datastructures::DelineatedDerivation< P > & deriv,
 datastructures::PolyRef poly)
```

```
0.14.33.2.29 poly_sgn_inv() template<typename P >
void smtrat::cadcells::operators::rules::poly_sgn_inv (
 datastructures::DelineatedDerivation< P > & deriv,
 datastructures::PolyRef poly)
```

```
0.14.33.2.30 root_ordering_holds() template<typename P >
void smtrat::cadcells::operators::rules::root_ordering_holds (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::IndexedRootOrdering & ordering)
```

```
0.14.33.2.31 root_ordering_holds_delineated() template<typename P >
bool smtrat::cadcells::operators::rules::root_ordering_holds_delineated (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::SymbolicInterval & ,
 const datastructures::IndexedRootOrdering & underlying_ordering,
 const datastructures::IndexedRootOrdering & ordering)
```

```
0.14.33.2.32 root_represents() template<typename P >
void smtrat::cadcells::operators::rules::root_represents (
 datastructures::SampledDerivation< P > & deriv,
 datastructures::IndexedRoot root)
```

```
0.14.33.2.33 root_well_def() template<typename P >
void smtrat::cadcells::operators::rules::root_well_def (
 datastructures::SampledDerivation< P > & deriv,
 datastructures::IndexedRoot root)
```

## 0.14.34 smtrat::cadcells::operators::rules::additional\_root\_outside\_util Namespace Reference

### Functions

- boost::container::flat\_set< datastructures::PolyRef > resultant\_polys (const datastructures::PolyRef &poly, const datastructures::IndexedRootOrdering &ordering)
- template<typename P >  
std::optional< datastructures::IndexedRoot > protect\_lower (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &ordering, const datastructures::PolyRef &poly, const boost::container::flat\_set< datastructures::PolyRef > &res\_polys, bool strict=false)

- template<typename P >  
`std::optional< datastructures::IndexedRoot > protect_upper (datastructures::SampledDerivation< P > &deriv, const datastructures::SymbolicInterval &cell, const datastructures::IndexedRootOrdering &ordering, const datastructures::PolyRef &poly, const boost::container::flat_set< datastructures::PolyRef > &res_polys, bool strict=false)`

#### 0.14.34.1 Function Documentation

**0.14.34.1.1 `protect_lower()`** template<typename P >  
`std::optional<datastructures::IndexedRoot> smtrat::cadcells::operators::rules::additional_<-`  
`root_outside_util::protect_lower (`  
 `datastructures::SampledDerivation< P > & deriv,`  
 `const datastructures::SymbolicInterval & cell,`  
 `const datastructures::IndexedRootOrdering & ordering,`  
 `const datastructures::PolyRef & poly,`  
 `const boost::container::flat_set< datastructures::PolyRef > & res_polys,`  
 `bool strict = false ) [inline]`

**0.14.34.1.2 `protect_upper()`** template<typename P >  
`std::optional<datastructures::IndexedRoot> smtrat::cadcells::operators::rules::additional_<-`  
`root_outside_util::protect_upper (`  
 `datastructures::SampledDerivation< P > & deriv,`  
 `const datastructures::SymbolicInterval & cell,`  
 `const datastructures::IndexedRootOrdering & ordering,`  
 `const datastructures::PolyRef & poly,`  
 `const boost::container::flat_set< datastructures::PolyRef > & res_polys,`  
 `bool strict = false ) [inline]`

**0.14.34.1.3 `resultant_polys()`** boost::container::flat\_set<datastructures::PolyRef> smtrat::cadcells<-  
`::operators::rules::additional_root_outside_util::resultant_polys (`  
 `const datastructures::PolyRef & poly,`  
 `const datastructures::IndexedRootOrdering & ordering ) [inline]`

### 0.14.35 smtrat::cadcells::operators::rules::filter\_util Namespace Reference

#### Enumerations

- enum class `result` {
 `NORMAL` , `INCLUSIVE` , `NORMAL_OPTIONAL` , `INCLUSIVE_OPTIONAL` ,  
`OMIT` }

#### Functions

- template<typename P >  
`void pseudo_order_invariant (datastructures::SampledDerivation< P > &deriv, const datastructures::PolyRef poly, const boost::container::flat_set< datastructures::PolyRef > &considered_polys)`
- template<typename P >  
`std::optional< carl::Interval< RAN > > delineable_interval (datastructures::Projections &proj, const Assignment &sample, const std::vector< datastructures::PolyRef > &polys)`
- template<typename P >  
`void filter_roots (datastructures::DelineatedDerivation< P > &deriv, const datastructures::PolyRef poly, std::function< result(const RAN &) > filter_condition)`
- template<typename P >  
`auto projection_root (const datastructures::DelineatedDerivation< P > &deriv, const RAN &root)`

- bool `has_common_real_root(datastructures::Projections &proj, Assignment ass, const datastructures::PolyRef &poly1, const datastructures::PolyRef &poly2)`

#### 0.14.35.1 Enumeration Type Documentation

##### 0.14.35.1.1 `result` enum `smtrat::cadcells::operators::rules::filter_util::result` [strong]

Enumerator

|                    |
|--------------------|
| NORMAL             |
| INCLUSIVE          |
| NORMAL_OPTIONAL    |
| INCLUSIVE_OPTIONAL |
| OMIT               |

#### 0.14.35.2 Function Documentation

```
0.14.35.2.1 delineable_interval() template<typename P >
std::optional<carl::Interval<RAN> > smtrat::cadcells::operators::rules::filter_util::delineable<-
_interval (
 datastructures::Projections & proj,
 const Assignment & sample,
 const std::vector< datastructures::PolyRef > & polys) [inline]
```

```
0.14.35.2.2 filter_roots() template<typename P >
void smtrat::cadcells::operators::rules::filter_util::filter_roots (
 datastructures::DelineatedDerivation< P > & deriv,
 const datastructures::PolyRef poly,
 std::function< result(const RAN &) > filter_condition) [inline]
```

```
0.14.35.2.3 has_common_real_root() bool smtrat::cadcells::operators::rules::filter_util::has<-
_common_real_root (
 datastructures::Projections & proj,
 Assignment ass,
 const datastructures::PolyRef & poly1,
 const datastructures::PolyRef & poly2) [inline]
```

```
0.14.35.2.4 projection_root() template<typename P >
auto smtrat::cadcells::operators::rules::filter_util::projection_root (
 const datastructures::DelineatedDerivation< P > & deriv,
 const RAN & root) [inline]
```

```
0.14.35.2.5 pseudo_order_invariant() template<typename P >
void smtrat::cadcells::operators::rules::filter_util::pseudo_order_invariant (
 datastructures::SampledDerivation< P > & deriv,
 const datastructures::PolyRef poly,
 const boost::container::flat_set< datastructures::PolyRef > & considered_polys)
[inline]
```

## 0.14.36 smtrat::cadcells::operators::rules::ordering\_util Namespace Reference

### Typedefs

- using `Decomposition` = `boost::container::flat_map<std::pair<datastructures::PolyRef, datastructures::PolyRef>, std::vector<datastructures::IndexedRootRelation>>`

### Functions

- void `add_to_decomposition` (`Decomposition &result, datastructures::PolyRef p1, datastructures::PolyRef p2, const datastructures::IndexedRootRelation &rel)`
- auto `decompose` (`const datastructures::IndexedRootOrdering &ordering`)

#### 0.14.36.1 Typedef Documentation

```
0.14.36.1.1 Decomposition using smtrat::cadcells::operators::rules::ordering_util::Decomposition
= typedef boost::container::flat_map<std::pair<datastructures::PolyRef, datastructures::PolyRef>, std::vector<datastructures::IndexedRootRelation>>
```

#### 0.14.36.2 Function Documentation

```
0.14.36.2.1 add_to_decomposition() void smtrat::cadcells::operators::rules::ordering_util::add_to_decomposition (
 Decomposition & result,
 datastructures::PolyRef p1,
 datastructures::PolyRef p2,
 const datastructures::IndexedRootRelation & rel) [inline]
```

```
0.14.36.2.2 decompose() auto smtrat::cadcells::operators::rules::ordering_util::decompose (
 const datastructures::IndexedRootOrdering & ordering) [inline]
```

## 0.14.37 smtrat::cadcells::representation Namespace Reference

Heuristics for computing representations.

### Namespaces

- `approximation`
- `util`

### Data Structures

- struct `cell`

*Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds.*
- struct `covering`
- struct `cell< CellHeuristic::BIGGEST_CELL_APPROXIMATION >`
- struct `cell< CellHeuristic::BIGGEST_CELL >`
- struct `cell< CellHeuristic::BIGGEST_CELL_EW >`
- struct `cell< CellHeuristic::CHAIN_EQ >`
- struct `cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ >`
- struct `cell< CellHeuristic::LOWEST_DEGREE_BARRIERS >`
- struct `cell< CellHeuristic::LOWEST_DEGREE_BARRIERS_EW >`
- struct `covering< CoveringHeuristic::BIGGEST_CELL_COVERING >`

- struct `covering< CoveringHeuristic::BIGGEST_CELL_COVERING_EW >`
- struct `covering< CoveringHeuristic::CHAIN_COVERING >`

## Typedefs

- using `IR = datastructures::IndexedRoot`

## Enumerations

- enum `CellHeuristic {  
 BIGGEST_CELL , CHAIN_EQ , LOWEST_DEGREE_BARRIERS , LOWEST_DEGREE_BARRIERS_EQ ,  
 BIGGEST_CELL_EW , LOWEST_DEGREE_BARRIERS_EW , BIGGEST_CELL_APPROXIMATION }`
- enum `CoveringHeuristic { BIGGEST_CELL_COVERING , CHAIN_COVERING , BIGGEST_CELL_COVERING_EW }`

## Functions

- `std::ostream & operator<< (std::ostream &os, CellHeuristic heuristic)`
- `std::ostream & operator<< (std::ostream &os, CoveringHeuristic heuristic)`
- template<typename T>  
`void compute_section_all_equational (datastructures::SampledDerivationRef< T > &der, datastructures::CellRepresentation< T > &response)`
- template<typename T>  
`void maintain_connectedness (datastructures::SampledDerivationRef< T > &der, datastructures::CellRepresentation< T > &response, bool enable_weak=false)`
- template<typename T>  
`void compute_barriers (datastructures::SampledDerivationRef< T > &der, datastructures::CellRepresentation< T > &response, bool section, bool enable_weak=false)`
- template<typename T>  
`std::vector< datastructures::SampledDerivationRef< T > > compute_min_ders (const std::vector< datastructures::SampledDerivationRef< T > > &ders)`
- template<typename T>  
`datastructures::IndexedRootOrdering compute_default_ordering (const std::vector< datastructures::CellRepresentation< T > > &cells, bool enable_weak=false)`

## Variables

- static const char \* `CellHeuristicStrings [] = { "BIGGEST_CELL", "CHAIN_EQ", "LOWEST_DEGREE_BARRIERS",  
 "LOWEST_DEGREE_BARRIERS_EQ", "BIGGEST_CELL_EW", "LOWEST_DEGREE_BARRIERS_EW" }`
- static const char \* `CoveringHeuristicStrings [] = { "BIGGEST_CELL_COVERING", "CHAIN_COVERING",  
 "BIGGEST_CELL_COVERING_EW" }`

### 0.14.37.1 Detailed Description

Heuristics for computing representations.

### 0.14.37.2 Typedef Documentation

#### 0.14.37.2.1 IR using `smtrat::cadcells::representation::IR = typedef datastructures::IndexedRoot`

#### 0.14.37.3 Enumeration Type Documentation

##### 0.14.37.3.1 CellHeuristic `enum smtrat::cadcells::representation::CellHeuristic`

Enumerator

|                            |
|----------------------------|
| BIGGEST_CELL               |
| CHAIN_EQ                   |
| LOWEST_DEGREE_BARRIERS     |
| LOWEST_DEGREE_BARRIERS_EQ  |
| BIGGEST_CELL_EW            |
| LOWEST_DEGREE_BARRIERS_EW  |
| BIGGEST_CELL_APPROXIMATION |

#### 0.14.37.3.2 **CoveringHeuristic** `enum smtrat::cadcells::representation::CoveringHeuristic`

Enumerator

|                          |
|--------------------------|
| BIGGEST_CELL_COVERING    |
| CHAIN_COVERING           |
| BIGGEST_CELL_COVERING_EW |

#### 0.14.37.4 Function Documentation

##### 0.14.37.4.1 **compute\_barriers()** `template<typename T >`

```
void smtrat::cadcells::representation::compute_barriers (
 datastructures::SampledDerivationRef< T > & der,
 datastructures::CellRepresentation< T > & response,
 bool section,
 bool enable_weak = false) [inline]
```

##### 0.14.37.4.2 **compute\_default\_ordering()** `template<typename T >`

```
datastructures::IndexedRootOrdering smtrat::cadcells::representation::compute_default_ordering (
(
 const std::vector< datastructures::CellRepresentation< T >> & cells,
 bool enable_weak = false)
```

##### 0.14.37.4.3 **compute\_min\_ders()** `template<typename T >`

```
std::vector<datastructures::SampledDerivationRef<T>> smtrat::cadcells::representation::compute_min_ders (
 const std::vector< datastructures::SampledDerivationRef< T >> & ders)
```

##### 0.14.37.4.4 **compute\_section\_all\_equational()** `template<typename T >`

```
void smtrat::cadcells::representation::compute_section_all_equational (
 datastructures::SampledDerivationRef< T > & der,
 datastructures::CellRepresentation< T > & response) [inline]
```

##### 0.14.37.4.5 **maintain\_connectedness()** `template<typename T >`

```
void smtrat::cadcells::representation::maintain_connectedness (
 datastructures::SampledDerivationRef< T > & der,
```

```
 datastructures::CellRepresentation< T > & response,
 bool enable_weak = false)
```

**0.14.37.4.6 operator<<()** [1/2] std::ostream& smtrat::cadcells::representation::operator<< ( std::ostream & os,  
CellHeuristic heuristic ) [inline]

**0.14.37.4.7 operator<<()** [2/2] std::ostream& smtrat::cadcells::representation::operator<< ( std::ostream & os,  
CoveringHeuristic heuristic ) [inline]

## 0.14.37.5 Variable Documentation

**0.14.37.5.1 CellHeuristicStrings** const char\* smtrat::cadcells::representation::CellHeuristicStrings[] = { "BIGGEST\_CELL", "CHAIN\_EQ", "LOWEST\_DEGREE\_BARRIERS", "LOWEST\_DEGREE\_BARRIERS\_EQ", "BIGGEST\_CELL\_EW", "LOWEST\_DEGREE\_BARRIERS\_EW" } [static]

**0.14.37.5.2 CoveringHeuristicStrings** const char\* smtrat::cadcells::representation::CoveringHeuristicStrings[] = { "BIGGEST\_CELL\_COVERING", "CHAIN\_COVERING", "BIGGEST\_CELL\_COVERING\_EW" } [static]

## 0.14.38 smtrat::cadcells::representation::approximation Namespace Reference

### Data Structures

- struct ApxSettings
- class CellApproximator
- class ApxCriteria

### TypeDefs

- using IR = datastructures::IndexedRoot

### Enumerations

- enum ApxPoly {  
SIMPLE , LINEAR\_GRADIENT , TAYLOR , TAYLOR\_LIN ,  
MAXIMIZE }
- enum ApxRoot { SAMPLE\_MID , SIMPLE\_REPRESENTATION , STERN\_BROCOT , FIXED\_RATIO }

### Functions

- const ApxSettings & apx\_settings ()
- template<> IR CellApproximator::apx\_bound< ApxPoly::SIMPLE > (const IR &, const RAN &bound, bool below)
- template<> IR CellApproximator::apx\_bound< ApxPoly::LINEAR\_GRADIENT > (const IR &p, const RAN &bound, bool below)
- template<> IR CellApproximator::apx\_bound< ApxPoly::TAYLOR > (const IR &p, const RAN &bound, bool below)
- template<> IR CellApproximator::apx\_bound< ApxPoly::TAYLOR\_LIN > (const IR &p, const RAN &bound, bool below)
- template<> IR CellApproximator::apx\_bound< ApxPoly::MAXIMIZE > (const IR &p, const RAN &bound, bool below)

- template<> `IR CellApproximator::apx_between< ApxPoly::SIMPLE >` (const `IR` &, const `IR` &, const `RAN` &l, const `RAN` &u)
- `Rational mediant (Rational a, Rational b)`
- `Rational approximate_RAN (const RAN &r)`
- `Rational approximate_RAN_sb (const RAN &r)`
- `Rational approximate_RAN_below (const RAN &r)`
- `Rational approximate_RAN_above (const RAN &r)`
- template<`ApxRoot AR`>
  - `Rational approximate_root_above (const RAN &inner, const RAN &outer)`
- template<`ApxRoot AR`>
  - `Rational approximate_root_below (const RAN &inner, const RAN &outer)`
- template<> `Rational approximate_root_above< ApxRoot::SAMPLE_MID >` (const `RAN` &inner, const `RAN` &outer)
- template<> `Rational approximate_root_below< ApxRoot::SAMPLE_MID >` (const `RAN` &inner, const `RAN` &outer)
- template<> `Rational approximate_root_above< ApxRoot::SIMPLE_REPRESENTATION >` (const `RAN` &inner, const `RAN` &outer)
- template<> `Rational approximate_root_below< ApxRoot::SIMPLE_REPRESENTATION >` (const `RAN` &inner, const `RAN` &outer)
- template<> `Rational approximate_root_above< ApxRoot::STERN_BROCOT >` (const `RAN` &inner, const `RAN` &outer)
- template<> `Rational approximate_root_below< ApxRoot::STERN_BROCOT >` (const `RAN` &inner, const `RAN` &outer)
- template<> `Rational approximate_root_above< ApxRoot::FIXED_RATIO >` (const `RAN` &inner, const `RAN` &outer)
- template<> `Rational approximate_root_below< ApxRoot::FIXED_RATIO >` (const `RAN` &inner, const `RAN` &outer)
- template<`ApxRoot AR`>
  - `Rational approximate_root (const RAN &inner, const RAN &outer, bool below)`

### 0.14.38.1 Typedef Documentation

**0.14.38.1.1 IR** `typedef datastructures::IndexedRoot smtrat::cadcells::representation::approximation::IR`

### 0.14.38.2 Enumeration Type Documentation

**0.14.38.2.1 ApxPoly** `enum smtrat::cadcells::representation::approximation::ApxPoly`

Enumerator

|                              |  |
|------------------------------|--|
| <code>SIMPLE</code>          |  |
| <code>LINEAR_GRADIENT</code> |  |
| <code>TAYLOR</code>          |  |
| <code>TAYLOR_LIN</code>      |  |
| <code>MAXIMIZE</code>        |  |

**0.14.38.2.2 ApxRoot** `enum smtrat::cadcells::representation::approximation::ApxRoot`

---

Enumerator

## Enumerator

|                       |
|-----------------------|
| SAMPLE_MID            |
| SIMPLE REPRESENTATION |
| STERN_BROCOOT         |
| FIXED_RATIO           |

**0.14.38.3 Function Documentation**

**0.14.38.3.1 approximate\_RAN()** `Rational smtrat::cadcells::representation::approximation::approximate<_RAN (const RAN & r) [inline]`

**0.14.38.3.2 approximate\_RAN\_above()** `Rational smtrat::cadcells::representation::approximation::approximate_RAN_above (const RAN & r) [inline]`

**0.14.38.3.3 approximate\_RAN\_below()** `Rational smtrat::cadcells::representation::approximation::approximate_RAN_below (const RAN & r) [inline]`

**0.14.38.3.4 approximate\_RAN\_sb()** `Rational smtrat::cadcells::representation::approximation::approximate_RAN_sb (const RAN & r) [inline]`

**0.14.38.3.5 approximate\_root()** `template<ApxRoot AR> Rational smtrat::cadcells::representation::approximation::approximate_root (const RAN & inner, const RAN & outer, bool below) [inline]`

**0.14.38.3.6 approximate\_root\_above()** `template<ApxRoot AR> Rational smtrat::cadcells::representation::approximation::approximate_root_above (const RAN & inner, const RAN & outer) [inline]`

**0.14.38.3.7 approximate\_root\_above< ApxRoot::FIXED\_RATIO >()** `template<> Rational smtrat::cadcells::representation::approximation::approximate_root_above< ApxRoot::FIXED_RATIO > (const RAN & inner, const RAN & outer) [inline]`

```
0.14.38.3.8 approximate_root_above< ApxRoot::SAMPLE_MID >() template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_above< ApxRoot:::↔
SAMPLE_MID > (
 const RAN & inner,
 const RAN & outer) [inline]
```

```
0.14.38.3.9 approximate_root_above< ApxRoot::SIMPLE REPRESENTATION >() template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_above< ApxRoot:::↔
SIMPLE REPRESENTATION > (
 const RAN & inner,
 const RAN & outer) [inline]
```

```
0.14.38.3.10 approximate_root_above< ApxRoot::STERN_BROCOT >() template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_above< ApxRoot:::↔
STERN_BROCOT > (
 const RAN & inner,
 const RAN & outer) [inline]
```

```
0.14.38.3.11 approximate_root_below() template<ApxRoot AR>
Rational smtrat::cadcells::representation::approximation::approximate_root_below (
 const RAN & inner,
 const RAN & outer)
```

```
0.14.38.3.12 approximate_root_below< ApxRoot::FIXED_RATIO >() template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_below< ApxRoot:::↔
FIXED_RATIO > (
 const RAN & inner,
 const RAN & outer) [inline]
```

```
0.14.38.3.13 approximate_root_below< ApxRoot::SAMPLE_MID >() template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_below< ApxRoot:::↔
SAMPLE_MID > (
 const RAN & inner,
 const RAN & outer) [inline]
```

```
0.14.38.3.14 approximate_root_below< ApxRoot::SIMPLE REPRESENTATION >() template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_below< ApxRoot:::↔
SIMPLE REPRESENTATION > (
 const RAN & inner,
 const RAN & outer) [inline]
```

```
0.14.38.3.15 approximate_root_below< ApxRoot::STERN_BROCOT >() template<>
Rational smtrat::cadcells::representation::approximation::approximate_root_below< ApxRoot:::↔
STERN_BROCOT > (
 const RAN & inner,
 const RAN & outer) [inline]
```

---

**0.14.38.3.16 `apx_settings()`** const `ApxSettings&` smtrat::cadcells::representation::approximation $\leftarrow$   
`::apx_settings ( ) [inline]`

**0.14.38.3.17 `CellApproximator::apx_between< ApxPoly::SIMPLE >()`** template $\langle\!\!\langle$   
`IR smtrat::cadcells::representation::approximation::CellApproximator::apx_between< ApxPoly::←`  
`SIMPLE > (`  
`const IR &,`  
`const IR &,`  
`const RAN & l,`  
`const RAN & u ) [inline]`

**0.14.38.3.18 `CellApproximator::apx_bound< ApxPoly::LINEAR_GRADIENT >()`** template $\langle\!\!\langle$   
`IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::←`  
`LINEAR_GRADIENT > (`  
`const IR & p,`  
`const RAN & bound,`  
`bool below ) [inline]`

**0.14.38.3.19 `CellApproximator::apx_bound< ApxPoly::MAXIMIZE >()`** template $\langle\!\!\langle$   
`IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::←`  
`MAXIMIZE > (`  
`const IR & p,`  
`const RAN & bound,`  
`bool below ) [inline]`

**0.14.38.3.20 `CellApproximator::apx_bound< ApxPoly::SIMPLE >()`** template $\langle\!\!\langle$   
`IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::←`  
`SIMPLE > (`  
`const IR &,`  
`const RAN & bound,`  
`bool below ) [inline]`

**0.14.38.3.21 `CellApproximator::apx_bound< ApxPoly::TAYLOR >()`** template $\langle\!\!\langle$   
`IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::←`  
`TAYLOR > (`  
`const IR & p,`  
`const RAN & bound,`  
`bool below ) [inline]`

**0.14.38.3.22 `CellApproximator::apx_bound< ApxPoly::TAYLOR_LIN >()`** template $\langle\!\!\langle$   
`IR smtrat::cadcells::representation::approximation::CellApproximator::apx_bound< ApxPoly::←`  
`TAYLOR_LIN > (`  
`const IR & p,`  
`const RAN & bound,`  
`bool below ) [inline]`

**0.14.38.3.23 `median()`** `Rational smtrat::cadcells::representation::approximation::median (`  
`Rational a,`  
`Rational b ) [inline]`

## 0.14.39 smtrat::cadcells::representation::util Namespace Reference

### Data Structures

- struct [PolyDelineation](#)
- struct [PolyDelineations](#)

### Functions

- bool [compare\\_simplest](#) ([datastructures::Projections](#) &proj, [datastructures::PolyRef](#) p1, [datastructures::PolyRef](#) p2)
- bool [max\\_degree](#) ([datastructures::Projections](#) &proj, [datastructures::RootFunction](#) rf)
- bool [compare\\_simplest](#) ([datastructures::Projections](#) &proj, [datastructures::RootFunction](#) rf1, [datastructures::RootFunction](#) rf2)
- std::optional< [datastructures::IndexedRoot](#) > [simplest\\_bound](#) ([datastructures::Projections](#) &proj, const std::vector< [datastructures::TaggedIndexedRoot](#) > &bounds, const boost::container::flat\_set< [datastructures::PolyRef](#) > &ignoring, bool enable\_weak=false)
- [datastructures::IndexedRoot](#) [simplest\\_bound](#) ([datastructures::Projections](#) &proj, const std::vector< [datastructures::TaggedIndexedRoot](#) > &bounds, bool enable\_weak=false)
- [datastructures::SymbolicInterval](#) [compute\\_simplest\\_cell](#) ([datastructures::Projections](#) &proj, const [datastructures::DelineationInterval](#) &delin, bool enable\_weak=false)
- std::optional< [datastructures::IndexedRootOrdering](#) > [simplest\\_biggest\\_cell\\_ordering](#) ([datastructures::Projections](#) &, [datastructures::Delineation](#) &delin, [datastructures::DelineationInterval](#) &delin\_interval, const [datastructures::SymbolicInterval](#) &interval, bool enable\_weak=false)
- std::optional< [datastructures::IndexedRootOrdering](#) > [simplest\\_chain\\_ordering](#) ([datastructures::Projections](#) &proj, [datastructures::Delineation](#) &delin, bool enable\_weak=false)
- void [decompose](#) (const [datastructures::Delineation](#) &delin, const [datastructures::DelineationInterval](#) &delin\_interval, [datastructures::Delineation](#) &delin\_out, [PolyDelineations](#) &poly\_delin\_out)
- void [add\\_chain\\_ordering](#) ([datastructures::IndexedRootOrdering](#) &out, const [datastructures::PolyRef](#) &poly, const [PolyDelineation](#) &poly\_delin)
- void [add\\_biggest\\_cell\\_ordering](#) ([datastructures::IndexedRootOrdering](#) &out, const [datastructures::PolyRef](#) &poly, const [PolyDelineation](#) &poly\_delin)
- void [add\\_weird\\_ordering](#) ([datastructures::IndexedRootOrdering](#) &out, const [datastructures::Delineation](#) &delin, const [datastructures::DelineationInterval](#) &delin\_interval, const [datastructures::SymbolicInterval](#) &interval)

### 0.14.39.1 Function Documentation

**0.14.39.1.1 add\_biggest\_cell\_ordering()** void smtrat::cadcells::representation::util::add\_biggest\_cell\_ordering (

```
 datastructures::IndexedRootOrdering & out,
 const datastructures::PolyRef & poly,
 const PolyDelineation & poly_delin) [inline]
```

**0.14.39.1.2 add\_chain\_ordering()** void smtrat::cadcells::representation::util::add\_chain\_ordering (

```
 datastructures::IndexedRootOrdering & out,
 const datastructures::PolyRef & poly,
 const PolyDelineation & poly_delin) [inline]
```

```
0.14.39.1.3 add_weird_ordering() void smtrat::cadcells::representation::util::add_weird_ordering (
 datastructures::IndexedRootOrdering & out,
 const datastructures::Delineation & delin,
 const datastructures::DelineationInterval & delin_interval,
 const datastructures::SymbolicInterval & interval) [inline]
```

  

```
0.14.39.1.4 compare_simplest() [1/2] bool smtrat::cadcells::representation::util::compare_simplest (
 datastructures::Projections & proj,
 datastructures::PolyRef p1,
 datastructures::PolyRef p2) [inline]
```

  

```
0.14.39.1.5 compare_simplest() [2/2] bool smtrat::cadcells::representation::util::compare_simplest (
 datastructures::Projections & proj,
 datastructures::RootFunction rf1,
 datastructures::RootFunction rf2) [inline]
```

  

```
0.14.39.1.6 compute_simplest_cell() datastructures::SymbolicInterval smtrat::cadcells::representation::util::compute_simplest_cell (
 datastructures::Projections & proj,
 const datastructures::DelineationInterval & del,
 bool enable_weak = false) [inline]
```

  

```
0.14.39.1.7 decompose() void smtrat::cadcells::representation::util::decompose (
 const datastructures::Delineation & delin,
 const datastructures::DelineationInterval & delin_interval,
 datastructures::Delineation & delin_out,
 PolyDelineations & poly_delin_out) [inline]
```

  

```
0.14.39.1.8 max_degree() bool smtrat::cadcells::representation::util::max_degree (
 datastructures::Projections & proj,
 datastructures::RootFunction rf) [inline]
```

  

```
0.14.39.1.9 simplest_biggest_cell_ordering() std::optional<datastructures::IndexedRootOrdering>
smtrat::cadcells::representation::util::simplest_biggest_cell_ordering (
 datastructures::Projections & ,
 datastructures::Delineation & delin,
 datastructures::DelineationInterval & delin_interval,
 const datastructures::SymbolicInterval & interval,
 bool enable_weak = false) [inline]
```

  

```
0.14.39.1.10 simplest_bound() [1/2] datastructures::IndexedRoot smtrat::cadcells::representation::util::simplest_bound (
 datastructures::Projections & proj,
 const std::vector< datastructures::TaggedIndexedRoot > & bounds,
 bool enable_weak = false) [inline]
```

```
0.14.39.1.11 simplest_bound() [2/2] std::optional<datastructures::IndexedRoot> smtrat::cadcells<->::representation::util::simplest_bound (
 datastructures::Projections & proj,
 const std::vector< datastructures::TaggedIndexedRoot > & bounds,
 const boost::container::flat_set< datastructures::PolyRef > & ignoring,
 bool enable_weak = false) [inline]
```

```
0.14.39.1.12 simplest_chain_ordering() std::optional<datastructures::IndexedRootOrdering> smtrat<->::cadcells::representation::util::simplest_chain_ordering (
 datastructures::Projections & proj,
 datastructures::Delineation & delin,
 bool enable_weak = false) [inline]
```

## 0.14.40 smtrat::compile\_information Namespace Reference

Compile time generated information about compiler and system version.

### Variables

- const std::string [SystemName](#) = "Linux"
- const std::string [SystemVersion](#) = "5.18.0-0.deb11.4-amd64"
- const std::string [BuildType](#) = "DEBUG"
- const std::string [CXXCompiler](#) = "/usr/bin/c++"
- const std::string [CXXCompilerVersion](#) = "11.3.0"
- const std::string [GitRevisionSHA1](#) = "f39a9ea3d790160bbca023015ac61529db908630"
- const std::string [PackageName](#) = "smtrat"
- const std::string [ProjectName](#) = "SMT-RAT"
- const std::string [Version](#) = "22.11"
- const std::string [Website](#) = "https://github.com/ths-rwth/smtrat/wiki"
- const std::string [GitVersion](#) = "22.11"

### 0.14.40.1 Detailed Description

Compile time generated information about compiler and system version.

### 0.14.40.2 Variable Documentation

**0.14.40.2.1 BuildType** const std::string smtrat::compile\_information::BuildType = "DEBUG"

**0.14.40.2.2 CXXCompiler** const std::string smtrat::compile\_information::CXXCompiler = "/usr/bin/c++"

**0.14.40.2.3 CXXCompilerVersion** const std::string smtrat::compile\_information::CXXCompiler<->Version = "11.3.0"

**0.14.40.2.4 GitRevisionSHA1** const std::string smtrat::compile\_information::GitRevisionSHA1 = "f39a9ea3d790160bbca023015ac61529db908630"

**0.14.40.2.5 GitVersion** const std::string smtrat::compile\_information::GitVersion = "22.11"

**0.14.40.2.6 PackageName** const std::string smrat::compile\_information::PackageName = "smrat"

**0.14.40.2.7 ProjectName** const std::string smrat::compile\_information::ProjectName = "SMT-RAT"

**0.14.40.2.8 SystemName** const std::string smrat::compile\_information::SystemName = "Linux"

**0.14.40.2.9 SystemVersion** const std::string smrat::compile\_information::SystemVersion = "5.18.0-0.deb11.4-amd64"

**0.14.40.2.10 Version** const std::string smrat::compile\_information::Version = "22.11"

**0.14.40.2.11 Website** const std::string smrat::compile\_information::Website = "https://github.com/thihs-rwth/smrat/wiki"

## 0.14.41 smrat::datastructures Namespace Reference

### Data Structures

- class [BinaryHeap](#)
- class [DynHeap](#)
- struct [OrderPair](#)
- struct [OrderPair< T1, T2, true >](#)

## 0.14.42 smrat::execution Namespace Reference

### Data Structures

- struct [Assertion](#)
- struct [SoftAssertion](#)
- struct [Objective](#)
- class [ExecutionState](#)

### Enumerations

- enum [Mode](#) { [START](#) =0 , [ASSERT](#) =1 , [SAT](#) =2 , [UNSAT](#) =3 }

### 0.14.42.1 Enumeration Type Documentation

**0.14.42.1.1 Mode** enum [smrat::execution::Mode](#)

#### Enumerator

|                        |  |
|------------------------|--|
| <a href="#">START</a>  |  |
| <a href="#">ASSERT</a> |  |
| <a href="#">SAT</a>    |  |
| <a href="#">UNSAT</a>  |  |

## 0.14.43 smtrat::expression Namespace Reference

### Namespaces

- [simplifier](#)

### Data Structures

- class [Expression](#)
- struct [ITEExpression](#)
- struct [QuantifierExpression](#)
- struct [UnaryExpression](#)
- struct [BinaryExpression](#)
- struct [NaryExpression](#)
- struct [ExpressionContent](#)
- struct [ExpressionTypeChecker](#)
- struct [ExpressionConverter](#)
- class [ExpressionPool](#)
- class [ExpressionVisitor](#)
- class [ExpressionModifier](#)

### Typedefs

- typedef std::vector< [Expression](#) > [Expressions](#)

### Enumerations

- enum [ITEType](#) { [ITE](#) }
- enum [QuantifierType](#) { [EXISTS](#) , [FORALL](#) }
- enum [UnaryType](#) { [NOT](#) }
- enum [BinaryType](#)
- enum [NaryType](#) { [AND](#) , [OR](#) , [XOR](#) , [IFF](#) }

### Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [Expression](#) &expr)
- std::ostream & [operator<<](#) (std::ostream &os, const [ITEExpression](#) &expr)
- std::ostream & [operator<<](#) (std::ostream &os, const [QuantifierExpression](#) &expr)
- std::ostream & [operator<<](#) (std::ostream &os, const [UnaryExpression](#) &expr)
- std::ostream & [operator<<](#) (std::ostream &os, const [BinaryExpression](#) &expr)
- std::ostream & [operator<<](#) (std::ostream &os, const [NaryExpression](#) &expr)
- std::ostream & [operator<<](#) (std::ostream &os, const [ExpressionContent](#) &expr)
- std::ostream & [operator<<](#) (std::ostream &os, const [ExpressionContent](#) \*expr)
- std::ostream & [operator<<](#) (std::ostream &os, [ITEType](#))
- std::ostream & [operator<<](#) (std::ostream &os, [QuantifierType](#) type)
- std::ostream & [operator<<](#) (std::ostream &os, [UnaryType](#) type)
- std::ostream & [operator<<](#) (std::ostream &os, [BinaryType](#) type)
- std::ostream & [operator<<](#) (std::ostream &os, [NaryType](#) type)

#### 0.14.43.1 Typedef Documentation

##### 0.14.43.1.1 Expressions `typedef std::vector<Expression> smtrat::expression::Expressions`

#### 0.14.43.2 Enumeration Type Documentation

**0.14.43.2.1 BinaryType** enum `smtrat::expression::BinaryType`**0.14.43.2.2 ITEType** enum `smtrat::expression::ITEType`

Enumerator

|     |                          |
|-----|--------------------------|
| ITE | <input type="checkbox"/> |
|-----|--------------------------|

**0.14.43.2.3 NaryType** enum `smtrat::expression::NaryType`

Enumerator

|     |                          |
|-----|--------------------------|
| AND | <input type="checkbox"/> |
| OR  | <input type="checkbox"/> |
| XOR | <input type="checkbox"/> |
| IFF | <input type="checkbox"/> |

**0.14.43.2.4 QuantifierType** enum `smtrat::expression::QuantifierType`

Enumerator

|        |                          |
|--------|--------------------------|
| EXISTS | <input type="checkbox"/> |
| FORALL | <input type="checkbox"/> |

**0.14.43.2.5 UnaryType** enum `smtrat::expression::UnaryType`

Enumerator

|     |                          |
|-----|--------------------------|
| NOT | <input type="checkbox"/> |
|-----|--------------------------|

**0.14.43.3 Function Documentation****0.14.43.3.1 operator<<()** [1/13] std::ostream& smtrat::expression::operator<< (  
    std::ostream & os,  
    BinaryType type) [inline]**0.14.43.3.2 operator<<()** [2/13] std::ostream& smtrat::expression::operator<< (  
    std::ostream & os,  
    const BinaryExpression & expr) [inline]**0.14.43.3.3 operator<<()** [3/13] std::ostream & smtrat::expression::operator<< (  
    std::ostream & os,  
    const Expression & expr)

**0.14.43.3.4 operator<<()** [4/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, const ExpressionContent & expr ) [inline]

**0.14.43.3.5 operator<<()** [5/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, const ExpressionContent \* expr ) [inline]

**0.14.43.3.6 operator<<()** [6/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, const ITEExpression & expr ) [inline]

**0.14.43.3.7 operator<<()** [7/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, const NaryExpression & expr ) [inline]

**0.14.43.3.8 operator<<()** [8/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, const QuantifierExpression & expr ) [inline]

**0.14.43.3.9 operator<<()** [9/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, const UnaryExpression & expr ) [inline]

**0.14.43.3.10 operator<<()** [10/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, ITEType ) [inline]

**0.14.43.3.11 operator<<()** [11/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, NaryType type ) [inline]

**0.14.43.3.12 operator<<()** [12/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, QuantifierType type ) [inline]

**0.14.43.3.13 operator<<()** [13/13] std::ostream& smtrat::expression::operator<< ( std::ostream & os, UnaryType type ) [inline]

## 0.14.44 smtrat::expression::simplifier Namespace Reference

### Data Structures

- struct [BaseSimplifier](#)
- struct [DuplicateSimplifier](#)
- struct [MergeSimplifier](#)

- struct `NegationSimplifier`
- struct `SimplifierChainCaller`
- struct `SimplifierChainCaller< 0 >`
- class `Simplifier`
- struct `SingletonSimplifier`

## Typedefs

- `typedef std::tuple< MergeSimplifier, DuplicateSimplifier, SingletonSimplifier > SimplifierChain`

### 0.14.44.1 Typedef Documentation

**0.14.44.1.1 SimplifierChain** `typedef std::tuple< MergeSimplifier, DuplicateSimplifier, SingletonSimplifier > smrat::expression::simplifier::SimplifierChain`

## 0.14.45 smrat::groebner Namespace Reference

### Typedefs

- `typedef std::map< carl::Variable, std::pair< TermT, carl::BitVector > > RewriteRules`

### Functions

- `template<typename Polynomial >  
static Polynomial rewritePolynomial (const Polynomial &inputPolynomial, const RewriteRules &rules)`

### 0.14.45.1 Typedef Documentation

**0.14.45.1.1 RewriteRules** `typedef std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smrat::groebner::RewriteRules`

### 0.14.45.2 Function Documentation

**0.14.45.2.1 rewritePolynomial()** `template<typename Polynomial >  
static Polynomial smrat::groebner::rewritePolynomial (const Polynomial & inputPolynomial,  
const RewriteRules & rules ) [static]`

## 0.14.46 smrat::helper Namespace Reference

### Functions

- `template<typename Pol >  
static size_t level_of (const VariableOrdering &order, const Pol &poly)`

### 0.14.46.1 Function Documentation

**0.14.46.1.1 level\_of()** `template<typename Pol >  
static size_t smrat::helper::level_of (const VariableOrdering & order,  
const Pol & poly ) [static]`

## 0.14.47 smtrat::icp Namespace Reference

### Data Structures

- class [ContractionCandidate](#)
- struct [contractionCandidateComp](#)
- class [ContractionCandidateManager](#)
- class [HistoryNode](#)
- class [IcpVariable](#)
- struct [icpVariableComp](#)

### Typedefs

- template<template< typename > class Operator>  
using [Contractor](#) = carl::Contraction< Operator, Poly >
- typedef [LRAVariable](#)< LRASettings1 >::[LRAVariable](#) [LRAVariable](#)
- typedef std::set< const [IcpVariable](#) \*, [icpVariableComp](#) > [set\\_icpVariable](#)

### Enumerations

- enum class [Updated](#) { [LEFT](#) , [RIGHT](#) , [BOTH](#) , [NONE](#) }

### Functions

- std::vector< Poly > [getNonlinearMonomials](#) (const Poly &\_expr)  
*Obtains the non-linear monomials of the given polynomial.*
- std::pair< ConstraintT, ConstraintT > [intervalToConstraint](#) (const Poly &\_lhs, const smtrat::DoubleInterval \_interval)  
*Creates a new constraint from an existing interval.*
- bool [intervalBoxContainsEmptyInterval](#) (const EvalDoubleIntervalMap &\_intervals)  
*Checks mIntervals if it contains an empty interval.*
- const LRAVariable \* [getOriginalLraVar](#) (carl::Variable::Arg \_var, const LRAVariable< LRASettings1 > &\_lra)

### 0.14.47.1 Typedef Documentation

**0.14.47.1.1 Contractor** template<template< typename > class Operator>  
using [smtrat::icp::Contractor](#) = typedef carl::Contraction<Operator, Poly>

**0.14.47.1.2 LRAVariable** typedef [LRAVariable](#)< LRASettings1 >::[LRAVariable](#) [smtrat::icp::LRAVariable](#)

**0.14.47.1.3 set\_icpVariable** typedef std::set<const [IcpVariable](#)\*, [icpVariableComp](#)> [smtrat::icp::set\\_icpVariable](#)

### 0.14.47.2 Enumeration Type Documentation

**0.14.47.2.1 Updated** enum [smtrat::icp::Updated](#) [strong]

#### Enumerator

|       |  |
|-------|--|
| LEFT  |  |
| RIGHT |  |
| BOTH  |  |
| NONE  |  |

### 0.14.47.3 Function Documentation

**0.14.47.3.1 getNonlinearMonomials()** `std::vector< Poly > smtrat::icp::getNonlinearMonomials ( const Poly & _expr )`

Obtains the non-linear monomials of the given polynomial.

#### Parameters

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <code>_expr</code> | The polynomial to obtain the non-linear monomials for. |
|--------------------|--------------------------------------------------------|

#### Returns

The non-linear monomials.

**0.14.47.3.2 getOriginalLraVar()** `const LRAModule< LRASettings1 >::LRAVariable * smtrat::icp::getOriginalLraVar (`

`carl::Variable::Arg _var,`  
`const LRAModule< LRASettings1 > & _lra )`

**0.14.47.3.3 intervalBoxContainsEmptyInterval()** `bool smtrat::icp::intervalBoxContainsEmptyInterval (`

`const EvalDoubleIntervalMap & _intervals )`

Checks mIntervals if it contains an empty interval.

#### Returns

**0.14.47.3.4 intervalToConstraint()** `std::pair< ConstraintT, ConstraintT > smtrat::icp::intervalToConstraint (`

`const Poly & _lhs,`  
`const smtrat::DoubleInterval _interval )`

Creates a new constraint from an existing interval.

#### Parameters

|                        |  |
|------------------------|--|
| <code>_interval</code> |  |
|------------------------|--|

#### Returns

`pair <lowerBoundConstraint*, upperBoundConstraint*>`

## 0.14.48 smtrat::impl Namespace Reference

### Data Structures

- class `fixedsize_allocator_impl`
- class `fixedsize_freelists`

*Publicly derive from all `fixedsize_freelist` types with chunk sizes between `SizeCurrent` and `SizeMax`.*

- class `fixedsize_freelists< SizeCurrent, SizeMax, true >`
- class `fixedsize_allocator_freelists`

- class [fixedsize\\_allocator\\_impl< T, SizeShift, true, true >](#)
- class [fixedsize\\_allocator\\_size\\_helper](#)

## Functions

- template<typename T >  
static CONSTEXPR std::size\_t [roundup\\_log2](#) (T)
- Used to find ceil(log2(n)) (the shift distance in the freelist allocator).*

## Variables

- static CONSTEXPR std::size\_t [LOWER\\_SIZE\\_BOUND](#) = sizeof(void\*)
- static CONSTEXPR std::size\_t [UPPER\\_SIZE\\_BOUND](#) = 256

### 0.14.48.1 Function Documentation

**0.14.48.1.1 roundup\_log2()** template<typename T >  
static CONSTEXPR std::size\_t smtrat::impl::roundup\_log2 (

T ) [inline], [static]

Used to find ceil(log2(n)) (the shift distance in the freelist allocator).

### 0.14.48.2 Variable Documentation

**0.14.48.2.1 LOWER\_SIZE\_BOUND** CONSTEXPR std::size\_t smtrat::impl::LOWER\_SIZE\_BOUND = sizeof(void\*)  
[static]

**0.14.48.2.2 UPPER\_SIZE\_BOUND** CONSTEXPR std::size\_t smtrat::impl::UPPER\_SIZE\_BOUND = 256  
[static]

## 0.14.49 smtrat::lra Namespace Reference

### Data Structures

- class [Variable](#)
- class [Bound](#)  
*Stores a bound, which could be an upper "<= b" or a lower bound ">= b" for a bound value b.*
- class [Numeric](#)
- class [TableauEntry](#)
- class [Tableau](#)
- class [Value](#)

### TypeDefs

- typedef size\_t [EntryID](#)

### Functions

- [Numeric abs](#) (const [Numeric](#) &\_value)  
*Calculates the absolute value of the given [Numeric](#).*
- [Numeric mod](#) (const [Numeric](#) &\_valueA, const [Numeric](#) &\_valueB)  
*Calculates the result of the first argument modulo the second argument.*
- [Numeric lcm](#) (const [Numeric](#) &\_valueA, const [Numeric](#) &\_valueB)

- **Numeric gcd** (const **Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Calculates the least common multiple of the two arguments.*
- **Numeric operator+** (const **Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Calculates the greatest common divisor of the two arguments.*
- **Numeric operator-** (const **Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Calculates the sum of the two given Numerics.*
- **Numeric operator-** (const **Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Calculates the difference between the two given Numerics.*
- **Numeric operator\*** (const **Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Calculates the product of the two given Numerics.*
- **Numeric operator/** (const **Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Calculates the division of the two given Numerics.*
- **Numeric & operator+=** (**Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Adds the value of the second given **Numeric** to the second given **Numeric**.*
- **Numeric & operator-=** (**Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Subtracts the second given **Numeric** to the first given **Numeric**.*
- **Numeric & operator\*=** (**Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Multiples the second given **Numeric** to the first given **Numeric**.*
- **Numeric & operator/=** (**Numeric** &\_valueA, const **Numeric** &\_valueB)
 

*Divides the first given **Numeric** by the second given **Numeric**.*
- **Numeric operator-** (const **Numeric** &\_value)
 

*Calculates the additive inverse of the given **Numeric**.*
- **Numeric & operator++** (**Numeric** &\_value)
 

*Increments the given **Numeric** by one.*
- **Numeric & operator--** (**Numeric** &\_value)
 

*Decrementes the given **Numeric** by one.*
- **std::ostream & operator<<** (**std::ostream** &\_out, const **Numeric** &\_value)
 

*Prints the given Numerics representation on the given output stream.*
- **template<typename T1> std::ostream & operator<<** (**std::ostream** &\_out, const **Value**< T1 > &\_value)
 

*Prints the given **Value** representation on the given output stream.*

## Variables

- static **EntryID** **LAST\_ENTRY\_ID** = 0

### 0.14.49.1 Typedef Documentation

#### 0.14.49.1.1 **EntryID** **typedef size\_t smtrat::lra::EntryID**

### 0.14.49.2 Function Documentation

#### 0.14.49.2.1 **abs()** **Numeric smtrat::lra::abs (const Numeric & \_value )**

Calculates the absolute value of the given **Numeric**.

#### Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <b>_value</b> | The <b>Numeric</b> to calculate the <b>Numeric</b> for. |
|---------------|---------------------------------------------------------|

**Returns**

The absolute value of the given [Numeric](#).

**0.14.49.2.2 gcd()** [Numeric](#) smtrat::lra::gcd (

```
const Numeric & _valueA,
const Numeric & _valueB)
```

Calculates the greatest common divisor of the two arguments.

Note, that this method can only be applied to integers.

**Parameters**

|                      |             |
|----------------------|-------------|
| <code>_valueA</code> | An integer. |
| <code>_valueB</code> | An integer. |

**Returns**

The least common divisor of the two arguments.

**0.14.49.2.3 lcm()** [Numeric](#) smtrat::lra::lcm (

```
const Numeric & _valueA,
const Numeric & _valueB)
```

Calculates the least common multiple of the two arguments.

Note, that this method can only be applied to integers.

**Parameters**

|                      |             |
|----------------------|-------------|
| <code>_valueA</code> | An integer. |
| <code>_valueB</code> | An integer. |

**Returns**

The least common multiple of the two arguments.

**0.14.49.2.4 mod()** [Numeric](#) smtrat::lra::mod (

```
const Numeric & _valueA,
const Numeric & _valueB)
```

Calculates the result of the first argument modulo the second argument.

Note, that this method can only be applied to integers.

**Parameters**

|                      |                  |
|----------------------|------------------|
| <code>_valueA</code> | An integer.      |
| <code>_valueB</code> | An integer != 0. |

**Returns**

The first argument modulo the second argument.

**0.14.49.2.5 operator\*()** [Numeric](#) smtrat::lra::operator\* (

```
 const Numeric & _valueA,
 const Numeric & _valueB)
```

Calculates the product of the two given Numerics.

#### Parameters

|         |                    |
|---------|--------------------|
| _valueA | The first factor.  |
| _valueB | The second factor. |

#### Returns

The product of the two given Numerics.

**0.14.49.2.6 operator\*=( )** `Numeric & smtrat::lra::operator*=(`  
 `Numeric & _valueA,`  
 `const Numeric & _valueB )`

Multiplies the second given [Numeric](#) to the first given [Numeric](#).

#### Parameters

|         |                                             |
|---------|---------------------------------------------|
| _valueA | The <a href="#">Numeric</a> to multiply.    |
| _valueB | The <a href="#">Numeric</a> to multiply by. |

#### Returns

The first given [Numeric](#) multiplied by the second given [Numeric](#).

**0.14.49.2.7 operator+()** `Numeric smtrat::lra::operator+ (`  
 `const Numeric & _valueA,`  
 `const Numeric & _valueB )`

Calculates the sum of the two given Numerics.

#### Parameters

|         |                     |
|---------|---------------------|
| _valueA | The first summand.  |
| _valueB | The second summand. |

#### Returns

The sum of the two given Numerics.

**0.14.49.2.8 operator++()** `Numeric & smtrat::lra::operator++ (`  
 `Numeric & _value )`

Increments the given [Numeric](#) by one.

#### Parameters

|        |                                           |
|--------|-------------------------------------------|
| _value | The <a href="#">Numeric</a> to increment. |
|--------|-------------------------------------------|

**Returns**

The given [Numeric](#) incremented by one.

**0.14.49.2.9 operator+=() [Numeric](#) & smtrat::lra::operator+= (**

```
 Numeric & _valueA,
 const Numeric & _valueB)
```

Adds the value of the second given [Numeric](#) to the second given [Numeric](#).

**Parameters**

|                         |                                        |
|-------------------------|----------------------------------------|
| <a href="#">_valueA</a> | The <a href="#">Numeric</a> to add to. |
| <a href="#">_valueB</a> | The <a href="#">Numeric</a> to add.    |

**Returns**

The first given [Numeric](#) increased by the second given [Numeric](#).

**0.14.49.2.10 operator-() [1/2] [Numeric](#) smtrat::lra::operator- (**

```
 const Numeric & _value)
```

Calculates the additive inverse of the given [Numeric](#).

**Parameters**

|                        |                                                                    |
|------------------------|--------------------------------------------------------------------|
| <a href="#">_value</a> | The <a href="#">Numeric</a> to calculate the additive inverse for. |
|------------------------|--------------------------------------------------------------------|

**Returns**

The additive inverse of the given [Numeric](#).

**0.14.49.2.11 operator-() [2/2] [Numeric](#) smtrat::lra::operator- (**

```
 const Numeric & _valueA,
 const Numeric & _valueB)
```

Calculates the difference between the two given Numerics.

**Parameters**

|                         |                 |
|-------------------------|-----------------|
| <a href="#">_valueA</a> | The minuend.    |
| <a href="#">_valueB</a> | The subtrahend. |

**Returns**

The difference between the two given Numerics.

**0.14.49.2.12 operator--() [Numeric](#) & smtrat::lra::operator-- (**

```
 Numeric & _value)
```

Decrements the given [Numeric](#) by one.

**Parameters**

|                        |                                           |
|------------------------|-------------------------------------------|
| <a href="#">_value</a> | The <a href="#">Numeric</a> to decrement. |
|------------------------|-------------------------------------------|

**Returns**

The given [Numeric](#) decremented by one.

**0.14.49.2.13 operator=( )** [Numeric](#) & smtrat::lra::operator== (

```
 Numeric & _valueA,
 const Numeric & _valueB)
```

Subtracts the second given [Numeric](#) to the first given [Numeric](#).

**Parameters**

|                         |                                               |
|-------------------------|-----------------------------------------------|
| <a href="#">_valueA</a> | The <a href="#">Numeric</a> to subtract from. |
| <a href="#">_valueB</a> | The <a href="#">Numeric</a> to subtract.      |

**Returns**

The first given [Numeric](#) subtracted by the second given [Numeric](#).

**0.14.49.2.14 operator/()** [Numeric](#) smtrat::lra::operator/ (

```
 const Numeric & _valueA,
 const Numeric & _valueB)
```

Calculates the division of the two given Numerics.

**Parameters**

|                         |               |
|-------------------------|---------------|
| <a href="#">_valueA</a> | The dividend. |
| <a href="#">_valueB</a> | The divisor.  |

**Returns**

The difference of the two given Numerics.

**0.14.49.2.15 operator/=( )** [Numeric](#) & smtrat::lra::operator/= (

```
 Numeric & _valueA,
 const Numeric & _valueB)
```

Divides the first given [Numeric](#) by the second given [Numeric](#).

**Parameters**

|                         |                                           |
|-------------------------|-------------------------------------------|
| <a href="#">_valueA</a> | The <a href="#">Numeric</a> to divide.    |
| <a href="#">_valueB</a> | The <a href="#">Numeric</a> to divide by. |

**Returns**

The first given [Numeric](#) divided by the second given [Numeric](#).

**0.14.49.2.16 operator<<()** [1/2] std::ostream & smtrat::lra::operator<< (

```
 std::ostream & _out,
 const Numeric & _value)
```

Prints the given Numerics representation on the given output stream.

**Parameters**

|                     |                                       |
|---------------------|---------------------------------------|
| <code>_out</code>   | The output stream to print on.        |
| <code>_value</code> | The <a href="#">Numeric</a> to print. |

**Returns**

The output stream after printing the given Numerics representation on it.

```
0.14.49.2.17 operator<<() [2/2] template<typename T1>
std::ostream& smtrat::lra::operator<<(
 std::ostream & _out,
 const Value< T1 > & _value)
```

**0.14.49.3 Variable Documentation**

**0.14.49.3.1 LAST\_ENTRY\_ID** [EntryID](#) smtrat::lra::LAST\_ENTRY\_ID = 0 [static]

**0.14.50 smtrat::lve Namespace Reference****Functions**

- [Rational evaluate](#) (carl::Variable v, const Poly &p, const Rational &r)
- [carl::Sign sgn](#) (carl::Variable v, const Poly &p, const RAN &r)
- [std::optional< ModelValue > get\\_root](#) (carl::Variable v, const Poly &p)
- [ModelValue get\\_non\\_root](#) (carl::Variable v, const Poly &p)
- [std::optional< ModelValue > get\\_value\\_for\\_sgn](#) (carl::Variable v, const Poly &p, carl::Sign sign)
- [carl::Sign sgn\\_of\\_invariant\\_poly](#) (carl::Variable v, const Poly &p)

**0.14.50.1 Function Documentation**

**0.14.50.1.1 evaluate()** [Rational](#) smtrat::lve::evaluate (

```
carl::Variable v,
const Poly & p,
const Rational & r)
```

**0.14.50.1.2 get\_non\_root()** [ModelValue](#) smtrat::lve::get\_non\_root (

```
carl::Variable v,
const Poly & p)
```

**0.14.50.1.3 get\_root()** [std::optional<ModelValue>](#) smtrat::lve::get\_root (

```
carl::Variable v,
const Poly & p)
```

**0.14.50.1.4 get\_value\_for\_sgn()** [std::optional<ModelValue>](#) smtrat::lve::get\_value\_for\_sgn (

```
carl::Variable v,
const Poly & p,
carl::Sign sign)
```

```
0.14.50.1.5 sgn() carl::Sign smtrat::lve::sgn (
 carl::Variable v,
 const Poly & p,
 const RAN & r)
```

```
0.14.50.1.6 sgn_of_invariant_poly() carl::Sign smtrat::lve::sgn_of_invariant_poly (
 carl::Variable v,
 const Poly & p)
```

## 0.14.51 smtrat::maxsmt Namespace Reference

Contains strategy implementations for max SMT computations.

### Data Structures

- class MaxSMTBackend
- class MaxSMTBackend< Solver, MaxSMTStrategy::FU\_MALIK\_INCREMENTAL >
- class MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR\_SEARCH >
- class MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >

### 0.14.51.1 Detailed Description

Contains strategy implementations for max SMT computations.

## 0.14.52 smtrat::mcsat Namespace Reference

### Namespaces

- arithmetic
- constraint\_type
- fm

*A Fourier-Motzkin based backend.*

- icp
- nlsat
- onecell
- onecellcad
- smtaf
- variableordering
- vs

### Data Structures

- struct SequentialAssignment
- struct FastParallelExplanation

*This explanation executes all given explanation in parallel processes and waits for the fastest explanation, returning the fastest delivered explanation, terminating all other parallel processes.*

- struct FullParallelExplanation

*This explanation executes all given explanation parallel in multiple threads and waits until every single one has finished, returning the result of the first listed explanation.*

- struct ParallelExplanation
- struct SequentialExplanation
- class Bookkeeping

*Represent the trail, i.e.*

- class ClauseChain

*An explanation is either a single clause or a chain of clauses, satisfying the following properties:*

- class [MCSATBackend](#)
- struct [InformationGetter](#)
- struct [TheoryLevel](#)
- class [MCSATMixin](#)

## Typedefs

- typedef allocator< std::optional< [Explanation](#) >, managed\_shared\_memory::segment\_manager > [ShmemAllocator](#)
- typedef vector< std::optional< [Explanation](#) >, [ShmemAllocator](#) > [SharedVector](#)
- using [ModelValues](#) = std::vector< std::pair< carl::Variable, [ModelError](#) > >
- using [AssignmentOrConflict](#) = std::variant< [ModelValues](#), [FormulasT](#) >
- using [Explanation](#) = std::variant< [FormulaT](#), [ClauseChain](#) >

## Enumerations

- enum class [ConstraintType](#) { [Constant](#), [Assigned](#), [Univariate](#), [Unassigned](#) }
- This type categorizes constraints with respect to a given (partial) model and the next variable to be assigned.*
- enum class [VariableOrdering](#) { [GreedyMaxUnivariate](#), [FeatureBased](#), [FeatureBasedZ3](#), [FeatureBasedBrown](#) }

## Functions

- [FormulaT resolveExplanation](#) (const [Explanation](#) &expl)
- std::ostream & [operator<<](#) (std::ostream &os, const [Bookkeeping](#) &bk)
- [FormulaT \\_transformToImplicationChain](#) (const [FormulaT](#) &formula, const [Model](#) &model, [ClauseChain](#) &chain, bool withEquivalences)
- std::ostream & [operator<<](#) (std::ostream &stream, const [ClauseChain::Link](#) &link)
- std::ostream & [operator<<](#) (std::ostream &stream, const [ClauseChain](#) &chain)
- std::ostream & [operator<<](#) (std::ostream &os, [ConstraintType](#) ct)
- std::string [get\\_name](#) ([VariableOrdering](#) ordering)
- std::ostream & [operator<<](#) (std::ostream &os, [VariableOrdering](#) ordering)
- template<[VariableOrdering](#) vot, typename [Constraints](#)>  
std::vector< carl::Variable > [calculate\\_variable\\_order](#) ([const Constraints](#) &c)
- template<[VariableOrdering](#) vot>  
std::vector< carl::Variable > [calculate\\_variable\\_order](#) ([const std::vector< \[ConstraintT\]\(#\) >](#) &constraints)
- template<typename Settings>  
std::ostream & [operator<<](#) (std::ostream &os, const [MCSATBackend](#)< [Settings](#) > &backend)

### 0.14.52.1 Typedef Documentation

**0.14.52.1.1 AssignmentOrConflict** using [smtrat::mcsat::AssignmentOrConflict](#) = [typedef std::variant<ModelValues, FormulasT>](#)

**0.14.52.1.2 Explanation** using [smtrat::mcsat::Explanation](#) = [typedef std::variant<FormulaT, ClauseChain>](#)

**0.14.52.1.3 ModelValues** using [smtrat::mcsat::ModelValues](#) = [typedef std::vector<std::pair<carl::Variable, ModelValue> >](#)

**0.14.52.1.4 SharedVector** [typedef vector<std::optional<Explanation>, ShmemAllocator> smtrat::mcsat::SharedVector](#)

---

**0.14.52.1.5 ShmemAllocator** `typedef allocator<std::optional<Explanation>, managed_shared<memory::segment_manager> smtrat::mcsat::ShmemAllocator`

### 0.14.52.2 Enumeration Type Documentation

#### 0.14.52.2.1 ConstraintType enum `smtrat::mcsat::ConstraintType` [strong]

This type categorizes constraints with respect to a given (partial) model and the next variable to be assigned. Note that Constant implies Assigned.

##### Enumerator

|            |                                                                                                                                         |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Constant   |                                                                                                                                         |
| Assigned   | The constraint contains no variables.                                                                                                   |
| Univariate | The constraint is fully assigned.                                                                                                       |
| Unassigned | The constraint has a single unassigned variable being the next one. The constraint has an unassigned variable that is not the next one. |

#### 0.14.52.2.2 VariableOrdering enum `smtrat::mcsat::VariableOrdering` [strong]

##### Enumerator

|                     |  |
|---------------------|--|
| GreedyMaxUnivariate |  |
| FeatureBased        |  |
| FeatureBasedZ3      |  |
| FeatureBasedBrown   |  |

### 0.14.52.3 Function Documentation

**0.14.52.3.1 `_transformToImplicationChain()`** `FormulaT smtrat::mcsat::_transformToImplicationChain( const FormulaT & formula, const Model & model, ClauseChain & chain, bool withEquivalences )`

**0.14.52.3.2 `calculate_variable_order()` [1/2]** `template<VariableOrdering vot, typename Constraints > std::vector<carl::Variable> smtrat::mcsat::calculate_variable_order( const Constraints & c )`

**0.14.52.3.3 `calculate_variable_order()` [2/2]** `template<VariableOrdering vot> std::vector<carl::Variable> smtrat::mcsat::calculate_variable_order( const std::vector< ConstraintT > & constraints )`

**0.14.52.3.4 `get_name()`** `std::string smtrat::mcsat::get_name( VariableOrdering ordering ) [inline]`

---

**0.14.52.3.5 operator<<()** [1/6] std::ostream& smtrat::mcsat::operator<< ( std::ostream & os, const Bookkeeping & bk ) [inline]

**0.14.52.3.6 operator<<()** [2/6] template<typename Settings> std::ostream& smtrat::mcsat::operator<< ( std::ostream & os, const MCSATBackend<Settings> & backend )

**0.14.52.3.7 operator<<()** [3/6] std::ostream& smtrat::mcsat::operator<< ( std::ostream & os, ConstraintType ct ) [inline]

**0.14.52.3.8 operator<<()** [4/6] std::ostream& smtrat::mcsat::operator<< ( std::ostream & os, VariableOrdering ordering ) [inline]

**0.14.52.3.9 operator<<()** [5/6] std::ostream& smtrat::mcsat::operator<< ( std::ostream & stream, const ClauseChain & chain ) [inline]

**0.14.52.3.10 operator<<()** [6/6] std::ostream& smtrat::mcsat::operator<< ( std::ostream & stream, const ClauseChain::Link & link ) [inline]

**0.14.52.3.11 resolveExplanation()** FormulaT smtrat::mcsat::resolveExplanation ( const Explanation & expl ) [inline]

## 0.14.53 smtrat::mcsat::arithmetic Namespace Reference

### Data Structures

- struct [AssignmentFinder](#)
- class [AssignmentFinder\\_detail](#)
- class [AssignmentFinder\\_ctx](#)
- class [Covering](#)

*Semantics:* The space is divided into a number of intervals: (-oo,a][a,a](a,b)[b,b](b,oo) A bit is set if the constraints refutes the corresponding interval.

- class [RootIndexer](#)

### Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [Covering](#) &ri)
- template<typename RANT> std::ostream & [operator<<](#) (std::ostream &os, const [RootIndexer](#)<RANT> &ri)

### 0.14.53.1 Function Documentation

```
0.14.53.1.1 operator<<() [1/2] std::ostream& smtrat::mcsat::arithmetic::operator<< (
 std::ostream & os,
 const Covering & ri) [inline]
```

```
0.14.53.1.2 operator<<() [2/2] template<typename RANT >
std::ostream& smtrat::mcsat::arithmetic::operator<< (
 std::ostream & os,
 const RootIndexer< RANT > & ri) [inline]
```

## 0.14.54 smtrat::mcsat::constraint\_type Namespace Reference

### Functions

- template<typename T >  
`ConstraintType categorize` (const T &t, const Model &model, carl::Variable next)  
• template<typename T >  
`bool is_constant` (const T &t)  
*Checks whether the constraint is constant, i.e.*  
• template<typename T >  
`bool isAssigned` (const T &t, const Model &model)  
*Checks whether all variables contained in the constraint are assigned in the model.*  
• template<typename T >  
`bool is_univariate` (const T &t, const Model &model, carl::Variable next)  
*Checks whether the constraint contains only a single unassigned variable, and this is the next one.*  
• template<typename T >  
`bool isUnassigned` (const T &t, const Model &model, carl::Variable next)  
*Checks whether the constraint contains an unassigned variable that is not the next one.*

### 0.14.54.1 Function Documentation

```
0.14.54.1.1 categorize() template<typename T >
ConstraintType smtrat::mcsat::constraint_type::categorize (
 const T & t,
 const Model & model,
 carl::Variable next)
```

```
0.14.54.1.2 is_constant() template<typename T >
bool smtrat::mcsat::constraint_type::is_constant (
 const T & t)
```

Checks whether the constraint is constant, i.e.  
whether it contains no variables.

```
0.14.54.1.3 is_univariate() template<typename T >
bool smtrat::mcsat::constraint_type::is_univariate (
 const T & t,
 const Model & model,
 carl::Variable next)
```

Checks whether the constraint contains only a single unassigned variable, and this is the next one.

```
0.14.54.1.4 isAssigned() template<typename T >
bool smtrat::mcsat::constraint_type::isAssigned (
 const T & t,
 const Model & model)
```

Checks whether all variables contained in the constraint are assigned in the model.

In particular, a Constant constraint is also Assigned.

```
0.14.54.1.5 isUnassigned() template<typename T >
bool smtrat::mcsat::constraint_type::isUnassigned (
 const T & t,
 const Model & model,
 carl::Variable next)
```

Checks whether the constraint contains an unassigned variable that is not the next one.

## 0.14.55 smtrat::mcsat::fm Namespace Reference

A Fourier-Motzkin based backend.

### Data Structures

- struct [Bound](#)
- struct [ConflictGenerator](#)
- struct [DefaultComparator](#)

*Does not order anything.*
- struct [MaxSizeComparator](#)

*This heuristic chooses the explanation excluding the largest interval.*
- struct [MinSizeComparator](#)

*This heuristic chooses the explanation excluding the smallest interval.*
- struct [MinVarCountComparator](#)

*This heuristic tries to minimize the number of variables occuring in the explanation.*
- struct [DefaultSettings](#)
- struct [IgnoreCoreSettings](#)
- struct [Explanation](#)

### Functions

- bool [isSubset](#) (const carl::Variables &subset, const carl::Variables &superset)
- std::ostream & [operator<<](#) (std::ostream &os, const [Bound](#) &b)

#### 0.14.55.1 Detailed Description

A Fourier-Motzkin based backend.

Preprocessing of constraints:

The input is a constraint  $c: p*x \sim q$  which can be used as a bound on  $x$  with  $p,q$  multivariate polynomials. If  $x$  only occurs linearly in  $c$ , this decomposition is possible. If  $p$  is zero, then  $c$  is conflicting iff  $!(0 \sim q)$ . If this is the case, we can return  $(c \&& p=0) \rightarrow 0 \sim q$  as explanation. Otherwise, we evaluate  $c$  over the partial model and obtain  $x \sim r$ , where  $r$  is a rational. To properly perform the elimination step detailed below, we additionally store whether  $p$  is negative over the current assignment as a Boolean.

We store  $(c,p,q,r,n)$  for each bound.

FM elimination:

Given a lower bound  $l$  and an upper bound  $u$ , the elimination is as follows: Conflict if  $l.r \geq u.r$  (or strict, if both relations from  $c$  are weak)  $l.q * u.p < u.q * l.p$

If exactly one of  $u.p$  and  $l.p$  was found to be negative, the relation has to be inverted. If  $u.p$  or  $l.p$  are not constants, we additionally have to add a literal stating that their sign does not change.

For all bounds involved, we add  $b.p < 0$  resp.  $b.p > 0$  as side condition to the explanation.

Handling of "not equal":

For linear arithmetic, a bound  $i$  belonging to a constraint with relation  $\neq$  can be in conflict with

- a bound e for a constraint with = iff  $i.r == e.r$ , then we return  $i.c \&& e.c \rightarrow i.q * e.p != e.q * i.p$
- two bounds l, u with  $\geq$  resp.  $\leq$  as relation and  $i.r == l.r == u.r$ , then we return  $i.c \&& l.c \&& u.c \&& (l.q * u.p = u.q * l.p) \rightarrow l.q * i.p != i.q * l.p$

For all bounds b involved, we add  $b.p != 0$  as side condition to the explanation.

### 0.14.55.2 Function Documentation

**0.14.55.2.1 `isSubset()`** `bool smtrat::mcsat::fm::isSubset (`  
    `const carl::Variables & subset,`  
    `const carl::Variables & superset ) [inline]`

**0.14.55.2.2 `operator<<()`** `std::ostream& smtrat::mcsat::fm::operator<< (`  
    `std::ostream & os,`  
    `const Bound & b ) [inline]`

## 0.14.56 smtrat::mcsat::icp Namespace Reference

### Data Structures

- class [Dependencies](#)
- struct [Explanation](#)
- struct [QueueEntry](#)
- class [IntervalPropagation](#)

## 0.14.57 smtrat::mcsat::nlsat Namespace Reference

### Namespaces

- [helper](#)

### Data Structures

- struct [Explanation](#)
- class [ExplanationGenerator](#)

## 0.14.58 smtrat::mcsat::nlsat::helper Namespace Reference

### Functions

- **`FormulaT buildFormulaFromVC (VariableComparisonT &&vc)`**  
*Construct a formula representing a variable comparison.*
- template<typename MVRootParams>  
**`FormulaT buildEquality (carl::Variable var, const MVRootParams &mvp)`**  
*Construct an atomic formula representing a variable being equal to the given multivariate root.*
- template<typename MVRootParams>  
**`FormulaT buildBelow (carl::Variable var, const MVRootParams &mvp)`**  
*Construct an atomic formula representing a variable being less than the given multivariate root.*
- template<typename MVRootParams>  
**`FormulaT buildAbove (carl::Variable var, const MVRootParams &mvp)`**  
*Construct an atomic formula representing a variable being greater than the given multivariate root.*
- std::set< [ConstraintT](#) > **`convertToConstraints (std::vector< FormulaT > constraintAtoms)`**  
*Transform constraints represented as atomic formulas into the easier to use objects of the Constraint class.*

### 0.14.58.1 Function Documentation

**0.14.58.1.1 buildAbove()** template<typename MVRootParams >  
`FormulaT smtrat::mcsat::nlsat::helper::buildAbove (`  
 `carl::Variable var,`  
 `const MVRootParams & mvp )`

Construct an atomic formula representing a variable being greater than the given multivariate root.  
 "v > root(..)"

**0.14.58.1.2 buildBelow()** template<typename MVRootParams >  
`FormulaT smtrat::mcsat::nlsat::helper::buildBelow (`  
 `carl::Variable var,`  
 `const MVRootParams & mvp )`

Construct an atomic formula representing a variable being less than the given multivariate root.  
 "v < root(..)"

**0.14.58.1.3 buildEquality()** template<typename MVRootParams >  
`FormulaT smtrat::mcsat::nlsat::helper::buildEquality (`  
 `carl::Variable var,`  
 `const MVRootParams & mvp )`

Construct an atomic formula representing a variable being equal to the given multivariate root.  
 "v = root(..)"

**0.14.58.1.4 buildFormulaFromVC()** `FormulaT smtrat::mcsat::nlsat::helper::buildFormulaFromVC (`  
 `VariableComparisonT && vc ) [inline]`

Construct a formula representing a variable comparison.

Simplify to a regular constraint if possible.

**0.14.58.1.5 convertToConstraints()** `std::set<ConstraintT> smtrat::mcsat::nlsat::helper::convertToConstraints (`  
 `std::vector< FormulaT > constraintAtoms ) [inline]`

Transform constraints represented as atomic formulas into the easier to use objects of the Constraint class.

## 0.14.59 smtrat::mcsat::onecell Namespace Reference

### Data Structures

- struct [Explanation](#)
- struct [LDBSettings](#)
- struct [LDBFilteredAllSelectiveSettings](#)
- struct [BCSettings](#)
- struct [BCFilteredSettings](#)
- struct [BCFilteredAllSettings](#)
- struct [BCFilteredBoundsSettings](#)
- struct [BCFilteredSamplesSettings](#)
- struct [BCFilteredAllSelectiveSettings](#)
- struct [BCApproximationSettings](#)

### Typedefs

- using [Settings](#) = [LDBFilteredAllSelectiveSettings](#)

## Functions

- template<typename Settings >  
std::optional< std::vector< cadcells::Atom > > **oncell** (const std::vector< cadcells::Atom > &constraints,  
const cadcells::Polynomial::ContextType &context, const cadcells::Assignment &sample)  
*An MCSAT-style single cell explanation function.*

## Variables

- constexpr static bool **use\_approximation** = false

### 0.14.59.1 Typedef Documentation

**0.14.59.1.1 Settings** using `smtrat::mcsat::oncell::Settings` = `typedef LDBFilteredAllSelectiveSettings`

### 0.14.59.2 Function Documentation

**0.14.59.2.1 oncell()** template<typename Settings >  
std::optional<std::vector<cadcells::Atom>> smtrat::mcsat::oncell::oncell (  
    const std::vector< cadcells::Atom > & constraints,  
    const cadcells::Polynomial::ContextType & context,  
    const cadcells::Assignment & sample )

An MCSAT-style single cell explanation function.

A set of constraints is called infeasible w.r.t. an assignment if the defining polynomials are univariate under the sample and there does not exist a value for the unassigned variable that satisfies all constraints.

This method eliminates the unassigned variable using `get_level_covering` or `get_delineation` and then constructs a single cell in the assigned variables level by level.

#### Parameters

|                          |                                                                                                                                                             |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>constraints</code> | Atoms of the same level such that sample cannot be extended for the highest variable without making one atom false. Note that this property is not checked. |
| <code>sample</code>      | A sample such that all but the highest variable in constraints are assigned.                                                                                |

#### Returns

A set of constraints whose conjunction describes an unsatisfying cell that can be concluded from the input constraints.

### 0.14.59.3 Variable Documentation

**0.14.59.3.1 use\_approximation** constexpr static bool `smtrat::mcsat::oncell::use_approximation`  
= false [static], [constexpr]

## 0.14.60 smtrat::mcsat::oncellcad Namespace Reference

### Namespaces

- `levelwise`
- `recursive`

*OneCell Explanation.*

## Data Structures

- class [RealAlgebraicPoint](#)  
*Represent a multidimensional point whose components are algebraic reals.*
- struct [TagPoly](#)  
*Tagged Polynomials.*
- struct [Section](#)  
*Represent a cell's (closed-interval-boundary) component along th k-th axis.*
- struct [Sector](#)  
*Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.*
- class [OneCellCAD](#)

## Typedefs

- using [RANMap](#) = std::map< carl::Variable, [RAN](#) >
- using [CADCell](#) = std::vector< std::variant< [Sector](#), [Section](#) > >  
*Represent a single cell [brown15].*

## Enumerations

- enum class [InvarianceType](#) { [ORD\\_INV](#) , [SIGN\\_INV](#) }  
*Invariance Types.*

## Functions

- template<typename PolyType >  
`bool hasOnlyNonConstIrreducibles (const std::vector< PolyType > &polys)`
- template<typename PolyType >  
`bool isNonConstIrreducible (const PolyType &poly)`
- template<typename T >  
`bool hasUniqElems (const std::vector< T > &container)`
- template<typename T >  
`bool isSubset (const std::vector< T > &c1, const std::vector< T > &c2)`
- template<typename T >  
`std::vector< T > asVector (const std::set< T > s)`
- template<typename PolyType >  
`bool polyVarsAreAllInList (const std::vector< PolyType > &polys, const std::vector< carl::Variable > &variables)`
- template<typename Number >  
`bool operator==(RealAlgebraicPoint< Number > &lhs, RealAlgebraicPoint< Number > &rhs)`  
*Check if two RealAlgebraicPoints are equal.*
- template<typename Number >  
`std::ostream & operator<< (std::ostream &os, const RealAlgebraicPoint< Number > &r)`  
*Streaming operator for a RealAlgebraicPoint.*
- `std::ostream & operator<< (std::ostream &os, const InvarianceType &inv)`
- `bool operator< (const InvarianceType &l, const InvarianceType &r)`
- `std::ostream & operator<< (std::ostream &os, const TagPoly &p)`
- `bool operator==(const TagPoly &lhs, const TagPoly &rhs)`
- `std::ostream & operator<< (std::ostream &os, const std::vector< TagPoly > &polys)`
- `std::ostream & operator<< (std::ostream &os, const std::vector< std::vector< TagPoly > > &lvls)`
- `std::size_t getDegree (TagPoly p, carl::Variable v)`
- `std::optional< std::size_t > levelOf (const std::vector< carl::Variable > &variableOrder, const Poly &poly)`  
*Find the index of the highest variable (wrt.*
- `std::vector< TagPoly > nonConstIrreducibleFactors (std::vector< carl::Variable > variableOrder, Poly poly, InvarianceType tag)`

- void `appendOnCorrectLevel` (const `Poly` &poly, `InvarianceType` tag, std::vector< std::vector< TagPoly >> &polys, std::vector< carl::Variable > variableOrder)
- std::ostream & `operator<<` (std::ostream &os, const `Section` &s)
- std::ostream & `operator<<` (std::ostream &os, const `Sector` &s)
- std::ostream & `operator<<` (std::ostream &os, const `CADCell` &cell)
- std::vector< `Poly` > `asMultiPolys` (const std::vector< TagPoly > polys)
- bool `contains` (const std::vector< TagPoly > &polys, const `Poly` &poly)
- template<typename T>  
bool `contains` (const std::vector< T > &list, const T &elem)
- `MultivariateRootT asRootExpr` (carl::Variable rootVariable, `Poly` poly, std::size\_t rootIdx)
- `RealAlgebraicPoint< smrat::Rational > asRANPoint` (const mcsat::Bookkeeping &data)
- template<typename T>  
std::vector< T > `prefix` (const std::vector< T > vars, std::size\_t prefixSize)
- std::vector< std::pair< `Poly`, `Poly` > > `duplicateElimination` (std::vector< std::pair< `Poly`, `Poly` >> vec)
- void `duplicateElimination` (std::vector< `Poly` > &vec)
- `Poly discriminant` (const carl::Variable &mainVariable, const `Poly` &p)  
*Projection related utilities for onecellcad.*
- `Poly resultant` (const carl::Variable &mainVariable, const `Poly` &p1, const `Poly` &p2)
- `Poly leadcoefficient` (const carl::Variable &mainVariable, const `Poly` &p)
- void `addResultants` (std::vector< std::pair< `Poly`, `Poly` >> &resultants, std::vector< std::vector< TagPoly >> &polys, carl::Variable mainVar, const std::vector< carl::Variable > &variableOrder)
- std::size\_t `cellDimension` (const `CADCell` &cell, const std::size\_t uptoLevel)
- `CADCell fullSpaceCell` (std::size\_t cellComponentCount)
- bool `optimized_singleLevelFullProjection` (carl::Variable mainVar, size\_t currentLevel, std::vector< std::vector< TagPoly >> &projectionLevels, OneCellCAD &cad)
- void `singleLevelFullProjection` (std::vector< carl::Variable > &variableOrder, carl::Variable mainVar, size\_t currentLevel, std::vector< std::vector< TagPoly >> &projectionLevels)

#### 0.14.60.1 Typedef Documentation

**0.14.60.1.1 CADCell** using `smrat::mcsat::onecellcad::CADCell` = `typedef std::vector<std::variant<Sector, Section> >`

Represent a single cell [brown15].

A cell is a collection of boundary objects along each axis, called cell-components based on math. vectors and their components.

**0.14.60.1.2 RANMap** using `smrat::mcsat::onecellcad::RANMap` = `typedef std::map<carl::Variable, RAN>`

#### 0.14.60.2 Enumeration Type Documentation

**0.14.60.2.1 InvarianceType** enum `smrat::mcsat::onecellcad::InvarianceType` [strong]  
Invariance Types.

Enumerator

|          |  |
|----------|--|
| ORD_INV  |  |
| SIGN_INV |  |

### 0.14.60.3 Function Documentation

#### 0.14.60.3.1 **addResultants()** `void smtrat::mcsat::oncellcad::addResultants (`

```
 std::vector< std::pair< Poly, Poly >> & resultants,
 std::vector< std::vector< TagPoly >> & polys,
 carl::Variable mainVar,
 const std::vector< carl::Variable > & variableOrder) [inline]
```

#### 0.14.60.3.2 **appendOnCorrectLevel()** `void smtrat::mcsat::oncellcad::appendOnCorrectLevel (`

```
 const Poly & poly,
 InvarianceType tag,
 std::vector< std::vector< TagPoly >> & polys,
 std::vector< carl::Variable > variableOrder) [inline]
```

#### 0.14.60.3.3 **asMultiPolys()** `std::vector<Poly> smtrat::mcsat::oncellcad::asMultiPolys (`

```
 const std::vector< TagPoly > polys) [inline]
```

#### 0.14.60.3.4 **asRANPoint()** `RealAlgebraicPoint<smtrat::Rational> smtrat::mcsat::oncellcad::as←`

```
RANPoint (
 const mcsat::Bookkeeping & data) [inline]
```

#### 0.14.60.3.5 **asRootExpr()** `MultivariateRootT smtrat::mcsat::oncellcad::asRootExpr (`

```
 carl::Variable rootVariable,
 Poly poly,
 std::size_t rootIdx) [inline]
```

##### Parameters

|                           |                                                                                                                                                                               |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>rootVariable</code> | The variable with respect to which the roots are computed in the end. It will be replaced by the special unique root-variable " <code>_z</code> " common in root-expressions. |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 0.14.60.3.6 **asVector()** `template<typename T >`

```
std::vector<T> smtrat::mcsat::oncellcad::asVector (
 const std::set< T > s)
```

#### 0.14.60.3.7 **cellDimension()** `std::size_t smtrat::mcsat::oncellcad::cellDimension (`

```
 const CADCell & cell,
 const std::size_t uptoLevel) [inline]
```

##### Returns

Dimension of a hypercube to which the given cell is homeomorphic, i.e., count the number of sectors of the given Cell restricted to its first components (with index 0 to 'uptoLevel').

#### 0.14.60.3.8 **contains()** `[1/2] template<typename T >`

```
bool smtrat::mcsat::oncellcad::contains (
```

```
 const std::vector< T > & list,
 const T & elem)
```

**0.14.60.3.9 `contains()` [2/2]** `bool smtrat::mcsat::oncellcad::contains (`  
`const std::vector< TagPoly > & polys,`  
`const Poly & poly )` [inline]

**0.14.60.3.10 `discriminant()`** `Poly smtrat::mcsat::oncellcad::discriminant (`  
`const carl::Variable & mainVariable,`  
`const Poly & p )` [inline]

Projection related utilities for onecellcad.

**0.14.60.3.11 `duplicateElimination()` [1/2]** `void smtrat::mcsat::oncellcad::duplicateElimination (`  
`std::vector< Poly > & vec )` [inline]

**0.14.60.3.12 `duplicateElimination()` [2/2]** `std::vector<std::pair<Poly, Poly> > smtrat::mcsat::oncellcad::duplicateElimination (`  
`std::vector< std::pair< Poly, Poly >> vec )` [inline]

**0.14.60.3.13 `fullSpaceCell()`** `CADCell smtrat::mcsat::oncellcad::fullSpaceCell (`  
`std::size_t cellComponentCount )` [inline]

**0.14.60.3.14 `getDegree()`** `std::size_t smtrat::mcsat::oncellcad::getDegree (`  
`TagPoly p,`  
`carl::Variable v )` [inline]

#### Parameters

|                |                               |
|----------------|-------------------------------|
| <code>p</code> | Polynomial to get degree from |
| <code>v</code> | Rootvariable for degree calc  |

#### Returns

**0.14.60.3.15 `hasOnlyNonConstIrreducibles()`** `template<typename PolyType >`  
`bool smtrat::mcsat::oncellcad::hasOnlyNonConstIrreducibles (`  
`const std::vector< PolyType > & polys )`

**0.14.60.3.16 `hasUniqElems()`** `template<typename T >`  
`bool smtrat::mcsat::oncellcad::hasUniqElems (`  
`const std::vector< T > & container )`

**0.14.60.3.17 `isNonConstIrreducible()`** `template<typename PolyType >`  
`bool smtrat::mcsat::oncellcad::isNonConstIrreducible (`  
`const PolyType & poly )`

---

**0.14.60.3.18 `isSubset()`** template<typename T>  
`bool smtrat::mcsat::onecellcad::isSubset (`  
 `const std::vector< T > & c1,`  
 `const std::vector< T > & c2 )`

**0.14.60.3.19 `leadcoefficient()`** `Poly smtrat::mcsat::onecellcad::leadcoefficient (`  
 `const carl::Variable & mainVariable,`  
 `const Poly & p ) [inline]`

**0.14.60.3.20 `levelOf()`** `std::optional<std::size_t> smtrat::mcsat::onecellcad::levelOf (`  
 `const std::vector< carl::Variable > & variableOrder,`  
 `const Poly & poly ) [inline]`

Find the index of the highest variable (wrt.

the ordering in 'variableOrder') that occurs with positive degree in 'poly'. Although 'level' is a math concept that starts counting at 1 we start counting at 0 and represent "no level/variable" as `std::nullopt` because it simplifies using the level directly as an index into arrays or vectors. Examples:

- `polyLevel(2) == nullopt` wrt. any variable order
  - `polyLevel(0*x+2) == nullopt` wrt. any variable order
  - `polyLevel(x+2) == 0` wrt.  $[x < y < z]$
  - `polyLevel(x+2) == 1` wrt.  $[y < x < z]$
  - `polyLevel(x+2) == 2` wrt.  $[y < z < x]$
  - `polyLevel(x*y+2) == 1` wrt.  $[x < y < z]$  because of y
  - `polyLevel(x*y+2) == 1` wrt.  $[y < x < z]$  because of x
  - `polyLevel(x*y+2) == 2` wrt.  $[x < z < y]$  because of y
- Preconditions:
- 'variables(poly)' must be a subset of 'variableOrder'.

**0.14.60.3.21 `nonConstIrreducibleFactors()`** `std::vector<TagPoly> smtrat::mcsat::onecellcad::nonConstIrreducibleFactors (`  
 `std::vector< carl::Variable > variableOrder,`  
 `Poly poly,`  
 `InvarianceType tag ) [inline]`

**0.14.60.3.22 `operator<()`** `bool smtrat::mcsat::onecellcad::operator< (`  
 `const InvarianceType & l,`  
 `const InvarianceType & r ) [inline]`

**0.14.60.3.23 `operator<<()` [1/8]** `std::ostream& smtrat::mcsat::onecellcad::operator<< (`  
 `std::ostream & os,`  
 `const CADCell & cell ) [inline]`

**0.14.60.3.24 `operator<<()` [2/8]** `std::ostream& smtrat::mcsat::onecellcad::operator<< (`  
 `std::ostream & os,`  
 `const InvarianceType & inv ) [inline]`

```
0.14.60.3.25 operator<<() [3/8] template<typename Number >
std::ostream& smtrat::mcsat::oncellcad::operator<< (
 std::ostream & os,
 const RealAlgebraicPoint< Number > & r)
```

Streaming operator for a [RealAlgebraicPoint](#).

```
0.14.60.3.26 operator<<() [4/8] std::ostream& smtrat::mcsat::oncellcad::operator<< (
 std::ostream & os,
 const Section & s) [inline]
```

```
0.14.60.3.27 operator<<() [5/8] std::ostream& smtrat::mcsat::oncellcad::operator<< (
 std::ostream & os,
 const Sector & s) [inline]
```

```
0.14.60.3.28 operator<<() [6/8] std::ostream& smtrat::mcsat::oncellcad::operator<< (
 std::ostream & os,
 const std::vector< std::vector< TagPoly >> & lvs) [inline]
```

```
0.14.60.3.29 operator<<() [7/8] std::ostream& smtrat::mcsat::oncellcad::operator<< (
 std::ostream & os,
 const std::vector< TagPoly > & polys) [inline]
```

```
0.14.60.3.30 operator<<() [8/8] std::ostream& smtrat::mcsat::oncellcad::operator<< (
 std::ostream & os,
 const TagPoly & p) [inline]
```

```
0.14.60.3.31 operator==() [1/2] bool smtrat::mcsat::oncellcad::operator== (
 const TagPoly & lhs,
 const TagPoly & rhs) [inline]
```

```
0.14.60.3.32 operator==() [2/2] template<typename Number >
bool smtrat::mcsat::oncellcad::operator== (
 RealAlgebraicPoint< Number > & lhs,
 RealAlgebraicPoint< Number > & rhs)
```

Check if two [RealAlgebraicPoints](#) are equal.

```
0.14.60.3.33 optimized_singleLevelFullProjection() bool smtrat::mcsat::oncellcad::optimized_←
singleLevelFullProjection (
 carl::Variable mainVar,
 size_t currentLevel,
 std::vector< std::vector< TagPoly >> & projectionLevels,
 OneCellCAD & cad) [inline]
```

```
0.14.60.3.34 polyVarsAreAllInList() template<typename PolyType >
bool smtrat::mcsat::oncellcad::polyVarsAreAllInList (
 const std::vector< PolyType > & polys,
 const std::vector< carl::Variable > & variables)
```

```
0.14.60.3.35 prefix() template<typename T >
std::vector<T> smtrat::mcsat::onecellcad::prefix (
 const std::vector< T > vars,
 std::size_t prefixSize)
```

  

```
0.14.60.3.36 resultant() Poly smtrat::mcsat::onecellcad::resultant (
 const carl::Variable & mainVariable,
 const Poly & p1,
 const Poly & p2) [inline]
```

  

```
0.14.60.3.37 singleLevelFullProjection() void smtrat::mcsat::onecellcad::singleLevelFullProjection
(
 std::vector< carl::Variable > & variableOrder,
 carl::Variable mainVar,
 size_t currentLevel,
 std::vector< std::vector< TagPoly >> & projectionLevels) [inline]
```

## 0.14.61 smtrat::mcsat::onecellcad::levelwise Namespace Reference

### Data Structures

- struct [SectionHeuristic1](#)
- struct [SectionHeuristic2](#)
- struct [SectionHeuristic3](#)
- struct [SectorHeuristic1](#)
- struct [SectorHeuristic2](#)
- struct [SectorHeuristic3](#)
- struct [Explanation](#)
- class [LevelwiseCAD](#)

## 0.14.62 smtrat::mcsat::onecellcad::recursive Namespace Reference

OneCell [Explanation](#).

### Data Structures

- struct [CoverNullification](#)
- struct [DontCoverNullification](#)
- struct [NoHeuristic](#)
- struct [DegreeAscending](#)
- struct [DegreeDescending](#)
- struct [Explanation](#)
- class [RecursiveCAD](#)

### Enumerations

- enum class [ShrinkResult](#) { [SUCCESS](#) , [FAIL](#) }

### Functions

- void [setNullification](#) (bool n)
- std::ostream & [operator<<](#) (std::ostream &os, const [ShrinkResult](#) &s)

## Variables

- bool `cover_nullification`

### 0.14.62.1 Detailed Description

OneCell [Explanation](#).

**0.14.62.1.1 Rationale** To better understand the CAD implementation this document concisely explains important CAD terminology from the literature and high-level implementation design decisions. Low-level design decisions for structs and classes are found in the implementation files themselves.

**0.14.62.1.2 Universe** The n-dimensional space in which a CADCell exists. The universe has n axes (=std basis vectors), numbered 0 to n-1, and each axis is associated with one variable.

**0.14.62.1.3 Variables order** A variable order like x,y,z tell us that the universe is 3-dimensional, gives a name to the each coordinate axis (thus also defines an axis ordering), tells us how to interpret a point like (5,2,1), and allows us to define properties like "level" for points and polynomials and "cylindrical" for CADCells.

**0.14.62.1.4 Level of a point** Number of components that a point has but interpreted with respect to the first variables in a variable ordering. E.g., if the universe has 3 axes/variables like x,y,z in that (increasing) order, then a point of level 2 has exactly 2 components representing the coordinates for the first two variables, x and y.

**0.14.62.1.5 Level of a polynomial** The number/index of the highest variable, with respect to a variable ordering, that appears with a positive degree in a polynomial. E.g., if the universe has 3 axes/variables like x,y,z in that (increasing) order, then a polynomial of level 2 at most mentions the first 2 variables, x and y, and definitely mentions second variable y but no "higher" variable like z .

**0.14.62.1.6 CADCell** A "cylindric algebraic cell" exists in an n-dimensional vector space called a "universe". A cell is a subspace of that universe with a possibly lower dimension and is "cylindric" only with respect to a specific variable ordering, numbered 0 to n-1. A cell is "algebraic", because its boundaries are represented by polynomials. Along each axis a cell is bound by either an non-empty open interval, called a "sector" and represented by two polynomials, or by a closed point-interval, called a "section" and represented by one polynomial. E.g. a section along axis k, said to be of "level k", requires a polynomial instead of a fixed number, because the fixed number can vary depending on the position along the (lower dimensional) axes 0 to k-1. This is why a section of level k is represented by a multivariate polynomial of "level k" (uses only the first k variables in the variable ordering). If we replace the first k-1 variables in that polynomial by specific numbers, we are left with a univariate polynomial whose root is then the fixed boundary number along axis k (if that polynomial has multiple roots, we also need to specify precisely which one of those represents the fixed boundary number). A different position, i.e., a different variable replacement, yields a possibly different univariate polynomial with a different root.

**0.14.62.1.7 Level of a sector/section** The number of the axis (with respect to some variable ordering that defines the same axis ordering) for which a sector/section defines the bounds. This number is also the level of the polynomial(s) inside the sector/section.

**0.14.62.1.8 Open CADCell** A cell is "open" if it only consists of sectors, i.e., it is open along all axes and is therefore a subspace with the same dimension as its universe.

### 0.14.62.2 Enumeration Type Documentation

#### 0.14.62.2.1 ShrinkResult enum `smtrat::mcsat::onecellcad::recursive::ShrinkResult` [strong]

Enumerator

|         |  |
|---------|--|
| SUCCESS |  |
| FAIL    |  |

### 0.14.62.3 Function Documentation

```
0.14.62.3.1 operator<<() std::ostream& smtrat::mcsat::onecellcad::recursive::operator<< (
 std::ostream & os,
 const ShrinkResult & s) [inline]
```

```
0.14.62.3.2 setNullification() void smtrat::mcsat::onecellcad::recursive::setNullification (
 bool n) [inline]
```

### 0.14.62.4 Variable Documentation

```
0.14.62.4.1 cover_nullification bool smtrat::mcsat::onecellcad::recursive::cover_nullification
```

## 0.14.63 smtrat::mcsat::smtaf Namespace Reference

### Data Structures

- struct [AssignmentFinder](#)
- struct [DefaultSettings](#)
- class [AssignmentFinder\\_SMT](#)

### Typedefs

- using [VariablePos](#) = std::vector< carl::Variable >::const\_iterator
- using [VariableRange](#) = std::pair< [VariablePos](#), [VariablePos](#) >

### Functions

- bool [includes](#) (const [VariableRange](#) &superset, const carl::Variables &subset)

### 0.14.63.1 Typedef Documentation

```
0.14.63.1.1 VariablePos using smtrat::mcsat::smtaf::VariablePos = typedef std::vector<carl::Variable>::const_iterator
```

```
0.14.63.1.2 VariableRange using smtrat::mcsat::smtaf::VariableRange = typedef std::pair<VariablePos, VariablePos>
```

### 0.14.63.2 Function Documentation

```
0.14.63.2.1 includes() bool smtrat::mcsat::smtaf::includes (
 const VariableRange & superset,
 const carl::Variables & subset) [inline]
```

## 0.14.64 smtrat::mcsat::variableordering Namespace Reference

### Namespaces

- [detail](#)

## Data Structures

- struct `VariableIDs`

## Functions

- template<typename Constraints >  
std::vector< carl::Variable > `feature_based` (const Constraints &c)
- template<typename Constraints >  
std::vector< carl::Variable > `feature_based_z3` (const Constraints &c)
- template<typename Constraints >  
std::vector< carl::Variable > `feature_based_brown` (const Constraints &c)  
*According to <https://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf>.*
- template<typename Constraints >  
std::vector< carl::Variable > `greedy_max_univariate` (const Constraints &c)
- std::ostream & `operator<<` (std::ostream &os, const `VariableIDs` &vids)
- template<typename Constraints >  
`void gatherVariables` (carl::carlVariables &vars, const Constraints &constraints)

### 0.14.64.1 Function Documentation

**0.14.64.1.1 `feature_based()`** template<typename Constraints >  
std::vector<carl::Variable> smtrat::mcsat::variableordering::`feature_based` (  
const Constraints & c )

**0.14.64.1.2 `feature_based_brown()`** template<typename Constraints >  
std::vector<carl::Variable> smtrat::mcsat::variableordering::`feature_based_brown` (  
const Constraints & c )

According to <https://www.usna.edu/Users/cs/wcbrown/research/ISSAC04/handout.pdf>.

**0.14.64.1.3 `feature_based_z3()`** template<typename Constraints >  
std::vector<carl::Variable> smtrat::mcsat::variableordering::`feature_based_z3` (  
const Constraints & c )

**0.14.64.1.4 `gatherVariables()`** template<typename Constraints >  
void smtrat::mcsat::variableordering::`gatherVariables` (  
carl::carlVariables & vars,  
const Constraints & constraints )

**0.14.64.1.5 `greedy_max_univariate()`** template<typename Constraints >  
std::vector<carl::Variable> smtrat::mcsat::variableordering::`greedy_max_univariate` (  
const Constraints & c )

**0.14.64.1.6 `operator<<()`** std::ostream& smtrat::mcsat::variableordering::`operator<<` (  
std::ostream & os,  
const `VariableIDs` & vids ) [inline]

## 0.14.65 smtrat::mcsat::variableordering::detail Namespace Reference

### Data Structures

- struct [FeatureCollector](#)

*This class manages features that are used to valuate variables on objects.*

### Functions

- template<typename Constraints , typename Calculator >  
`double abstract_feature (const Constraints &constraints, double initial, std::function< double(double, double)> &&comb, Calculator &&calc)`
- template<typename Constraints >  
`double max_degree (const Constraints &constraints, carl::Variable v)`  
*The maximum degree of this variable.*
- template<typename Constraints >  
`double max_term_total_degree (const Constraints &constraints, carl::Variable v)`  
*The maximum total degree of a term involving this variable.*
- template<typename Constraints >  
`double max_coefficient (const Constraints &constraints, carl::Variable v)`
- template<typename Constraints >  
`double num_occurrences (const Constraints &constraints, carl::Variable v)`
- carl::Bitset `variablesOf (const ConstraintT &c, VariableIDs &vids)`
- long `countUnivariates (const std::vector< carl::Bitset > &constraints, std::size_t id)`
- bool `stillOccurs (const std::vector< carl::Bitset > &constraints, std::size_t id)`
- carl::Variable `findMax (const std::vector< carl::Bitset > &constraints, const VariableIDs &vids)`
- void `purgeVariable (std::vector< carl::Bitset > &constraints, carl::Variable v, const VariableIDs &vids)`

#### 0.14.65.1 Function Documentation

**0.14.65.1.1 `abstract_feature()`** template<typename Constraints , typename Calculator >  
`double smtrat::mcsat::variableordering::detail::abstract_feature (`  
 `const Constraints & constraints,`  
 `double initial,`  
 `std::function< double(double, double)> && comb,`  
 `Calculator && calc )`

**0.14.65.1.2 `countUnivariates()`** long `smtrat::mcsat::variableordering::detail::countUnivariates (`  
 `const std::vector< carl::Bitset > & constraints,`  
 `std::size_t id ) [inline]`

**0.14.65.1.3 `findMax()`** carl::Variable `smtrat::mcsat::variableordering::detail::findMax (`  
 `const std::vector< carl::Bitset > & constraints,`  
 `const VariableIDs & vids ) [inline]`

**0.14.65.1.4 `max_coefficient()`** template<typename Constraints >  
`double smtrat::mcsat::variableordering::detail::max_coefficient (`  
 `const Constraints & constraints,`  
 `carl::Variable v )`

```
0.14.65.1.5 max_degree() template<typename Constraints >
double smtrat::mcsat::variableordering::detail::max_degree (
 const Constraints & constraints,
 carl::Variable v)
```

The maximum degree of this variable.

```
0.14.65.1.6 max_term_total_degree() template<typename Constraints >
double smtrat::mcsat::variableordering::detail::max_term_total_degree (
 const Constraints & constraints,
 carl::Variable v)
```

The maximum total degree of a term involving this variable.

```
0.14.65.1.7 num_occurrences() template<typename Constraints >
double smtrat::mcsat::variableordering::detail::num_occurrences (
 const Constraints & constraints,
 carl::Variable v)
```

```
0.14.65.1.8 purgeVariable() void smtrat::mcsat::variableordering::detail::purgeVariable (
 std::vector< carl::Bitset > & constraints,
 carl::Variable v,
 const VariableIDs & vids) [inline]
```

```
0.14.65.1.9 stillOccurs() bool smtrat::mcsat::variableordering::detail::stillOccurs (
 const std::vector< carl::Bitset > & constraints,
 std::size_t id) [inline]
```

```
0.14.65.1.10 variablesOf() carl::Bitset smtrat::mcsat::variableordering::detail::variablesOf (
 const ConstraintT & c,
 VariableIDs & vids) [inline]
```

## 0.14.66 smtrat::mcsat::vs Namespace Reference

### Namespaces

- [helper](#)

### Data Structures

- struct [Explanation](#)
- struct [DefaultSettings](#)
- class [ExplanationGenerator](#)

## 0.14.67 smtrat::mcsat::vs::helper Namespace Reference

### Data Structures

- struct [TestCandidate](#)

### Functions

- void [getFormulaAtoms](#) (const [FormulaT](#) &f, [FormulaSetT](#) &result)
- [FormulaT to\\_formula](#) (const carl::vs::CaseDistinction< [Poly](#) > &docc)  
*Converts a DisjunctionOfConstraintConjunctions to a regular Formula.*

- static bool `generateZeros` (const `FormulaT` &formula, const `carl::Variable` &eliminationVar, std::function< void(`SqrtEx` &&`sqrteXpression`, `ConstraintsT` &&`sideConditions`)> `yield_result`)
 

*Get zeros with side conditions of the given constraint.*
- static bool `addOrMergeTestCandidate` (std::vector< `TestCandidate` > &results, const `TestCandidate` &newSubstitution)
 

*Adds a new substitution to the given list of substitutions or merges it to an existing one.*
- static bool `generateTestCandidates` (std::vector< `TestCandidate` > &results, const `carl::Variable` &eliminationVar, const `FormulaSetT` &constraints)
 

*Generate all test candidates according to "vanilla" virtual substitution.*
- bool `substitute` (const `FormulaT` &constr, const `carl::Variable` var, const `carl::vs::Term`< `Poly` > &term, `FormulaT` &result)
 

*Substitutes the variable var in constr by term.*

#### 0.14.67.1 Function Documentation

**0.14.67.1.1 `addOrMergeTestCandidate()`** static bool smtrat::mcsat::vs::helper::addOrMergeTestCandidate (
 std::vector< `TestCandidate` > & results,
 const `TestCandidate` & newSubstitution ) [static]

Adds a new substitution to the given list of substitutions or merges it to an existing one.

Returns true if a new substitution was created.

**0.14.67.1.2 `generateTestCandidates()`** static bool smtrat::mcsat::vs::helper::generateTestCandidates (
 std::vector< `TestCandidate` > & results,
 const `carl::Variable` & eliminationVar,
 const `FormulaSetT` & constraints ) [static]

Generate all test candidates according to "vanilla" virtual substitution.

Returns false iff VS is not applicable.

**0.14.67.1.3 `generateZeros()`** static bool smtrat::mcsat::vs::helper::generateZeros (
 const `FormulaT` & formula,
 const `carl::Variable` & eliminationVar,
 std::function< void(`SqrtEx` &&`sqrteXpression`, `ConstraintsT` &&`sideConditions`)>
 yield\_result ) [static]

Get zeros with side conditions of the given constraint.

Kind of a generator function. Passes generated zeros to a callback function to avoid copying.

**0.14.67.1.4 `getFormulaAtoms()`** void smtrat::mcsat::vs::helper::getFormulaAtoms (
 const `FormulaT` & f,
 `FormulaSetT` & result ) [inline]

**0.14.67.1.5 `substitute()`** bool smtrat::mcsat::vs::helper::substitute (
 const `FormulaT` & constr,
 const `carl::Variable` var,
 const `carl::vs::Term`< `Poly` > & term,
 `FormulaT` & result ) [inline]

**0.14.67.1.6 `to_formula()`** `FormulaT` smtrat::mcsat::vs::helper::to\_formula (
 const `carl::vs::CaseDistinction`< `Poly` > & docc ) [inline]

Converts a DisjunctionOfConstraintConjunctions to a regular Formula.

## 0.14.68 smtrat::onecellcad Namespace Reference

### Namespaces

- [recursive](#)

## 0.14.69 smtrat::onecellcad::recursive Namespace Reference

### Data Structures

- struct [Section](#)

*Represent a cell's closed-interval-boundary object along one single axis by an irreducible, multivariate polynomial of level k.*

- struct [Sector](#)

*Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.*

### Typedefs

- using [UniPoly](#) = carl::UnivariatePolynomial< [smtrat::Rational](#) >
- using [MultiPoly](#) = carl::MultivariatePolynomial< [smtrat::Rational](#) >
- using [MultiCoeffUniPoly](#) = carl::UnivariatePolynomial< [MultiPoly](#) >
- using [RANPoint](#) = RealAlgebraicPoint< [smtrat::Rational](#) >
- using [RANMap](#) = std::map< carl::Variable, [RAN](#) >
- using [OpenCADCell](#) = std::vector< [Sector](#) >

*Represent a single open cell [1].*

### Functions

- std::ostream & [operator<<](#) (std::ostream &o, const [Section](#) &s)
- std::ostream & [operator<<](#) (std::ostream &o, const [Sector](#) &s)
- std::ostream & [operator<<](#) (std::ostream &o, const [OpenCADCell](#) &c)
- [OpenCADCell](#) [createFullspaceCoveringCell](#) (size\_t level)
- size\_t [levelOf](#) (const [MultiPoly](#) &poly, const std::vector< carl::Variable > &variableOrder)  
*Find the index of the highest variable (wrt.*
- std::map< carl::Variable, [RAN](#) > [toStdMap](#) (const [RANPoint](#) &point, size\_t count, const std::vector< carl::Variable > &variableOrder)  
*Create a mapping from the first 'count'-many variables in the given 'variableOrder' to the first 'count'-many components of the given 'point'.*
- std::optional< [OpenCADCell](#) > [mergeCellWithPoly](#) ([OpenCADCell](#) &cell, const [RANPoint](#) &point, const std::vector< carl::Variable > &variableOrder, const [MultiPoly](#) poly)  
*Merge the given open [OpenCADCell](#) 'cell' that contains the given 'point' (called "alpha" in [1]) with a polynomial 'poly'.*
- std::optional< [OpenCADCell](#) > [createOpenCADCell](#) (const std::vector< [MultiPoly](#) > polySet, const [RANPoint](#) &point, const std::vector< carl::Variable > &variableOrder)  
*Construct a [OpenCADCell](#) for the given set of polynomials 'polySet' that contains the given 'point'.*

### 0.14.69.1 Typedef Documentation

**0.14.69.1.1 MultiCoeffUniPoly** using [smtrat::onecellcad::recursive::MultiCoeffUniPoly](#) = [typedef](#)  
carl::UnivariatePolynomial<[MultiPoly](#)>

**0.14.69.1.2 MultiPoly** using [smtrat::onecellcad::recursive::MultiPoly](#) = [typedef](#) carl::MultivariatePolynomial<[smtrat::Rational](#)>

**0.14.69.1.3 OpenCADCell** using `smtrat::onecellcad::recursive::OpenCADCell` = `typedef std::vector<Sector>`

Represent a single open cell [1].

A cell is a collection of boundary objects along each axis. In case of an open cell, the boundary objects are all sectors.

**0.14.69.1.4 RANMap** using `smtrat::onecellcad::recursive::RANMap` = `typedef std::map<carl::Variable, RAN>`

**0.14.69.1.5 RANPoint** using `smtrat::onecellcad::recursive::RANPoint` = `typedef RealAlgebraic::Point<smtrat::Rational>`

**0.14.69.1.6 UniPoly** using `smtrat::onecellcad::recursive::UniPoly` = `typedef carl::Univariate<Polynomial<smtrat::Rational>>`

## 0.14.69.2 Function Documentation

**0.14.69.2.1 createFullspaceCoveringCell()** `OpenCADCell smtrat::onecellcad::recursive::createFullspaceCoveringCell ( size_t level )`

**0.14.69.2.2 createOpenCADCell()** `std::optional<OpenCADCell> smtrat::onecellcad::recursive::createOpenCADCell ( const std::vector< MultiPoly > polySet, const RANPoint & point, const std::vector< carl::Variable > & variableOrder )`

Construct a OpenCADCell for the given set of polynomials 'polySet' that contains the given 'point'.

The returned cell represents the largest region that is sign-invariant on all polynomials in the 'polySet' and is cylindrical with respect to the variables ordering given in 'variableOrder'. Note that this construction is only correct if plugging in the 'point' into any polynomial of the 'polySet' yields a non-zero value, i.e. 'point' is not a root of any polynomial in 'polySet', otherwise no OpenCADCell is returned. Note that the dimension (number of components) of the 'point' == the number of variables in 'variableOrder' and that any polynomial of the 'polySet' must be irreducible and mention only variables from 'variableOrder'.

**0.14.69.2.3 levelOf()** `size_t smtrat::onecellcad::recursive::levelOf ( const MultiPoly & poly, const std::vector< carl::Variable > & variableOrder )`

Find the index of the highest variable (wrt.

the ordering in 'variableOrder') that occurs with positive degree in 'poly'. Since levelOf is a math concept it starts counting at 1! Examples:

- `levelOf(2) == 0` wrt. any variable order
- `levelOf(0*x+2) == 0` wrt. any variable order
- `levelOf(x+2) == 1` wrt.  $[x < y < z]$
- `levelOf(x+2) == 2` wrt.  $[y < x < z]$
- `levelOf(x+2) == 3` wrt.  $[y < z < x]$
- `levelOf(x*y+2) == 2` wrt.  $[x < y < z]$  because of y
- `levelOf(x*y+2) == 2` wrt.  $[y < x < z]$  because of x

- `levelOf(x*y+2) == 3` wrt.  $[x < z < y]$  because of y Preconditions:
- 'variables(poly)' must be a subset of 'variableOrder'.

#### 0.14.69.2.4 `mergeCellWithPoly()` `std::optional<OpenCADCell> smtrat::oncellcad::recursive::mergeCellWithPoly (`

```
 OpenCADCell & cell,
 const RANPoint & point,
 const std::vector< carl::Variable > & variableOrder,
 const MultiPoly poly)
```

Merge the given open OpenCADCell 'cell' that contains the given 'point' (called "alpha" in [1]) with a polynomial 'poly'.

Before the merge 'cell' represents a region that is sign-invariant on other (previously merged) polynomials (all signs are non-zero). The returned cell represents a region that is additionally sign-invariant on 'poly' (also with non-zero sign).

##### Returns

either an OpenCADCell or nothing (representing a failed construction)

#### 0.14.69.2.5 `operator<<()` [1/3] `std::ostream& smtrat::oncellcad::recursive::operator<< (`

```
 std::ostream & o,
 const OpenCADCell & c)
```

#### 0.14.69.2.6 `operator<<()` [2/3] `std::ostream& smtrat::oncellcad::recursive::operator<< (`

```
 std::ostream & o,
 const Section & s)
```

#### 0.14.69.2.7 `operator<<()` [3/3] `std::ostream& smtrat::oncellcad::recursive::operator<< (`

```
 std::ostream & o,
 const Sector & s)
```

#### 0.14.69.2.8 `toStdMap()` `std::map<carl::Variable, RAN> smtrat::oncellcad::recursive::toStdMap (`

```
 const RANPoint & point,
 size_t count,
 const std::vector< carl::Variable > & variableOrder)
```

Create a mapping from the first 'count'-many variables in the given 'variableOrder' to the first 'count'-many components of the given 'point'.

## 0.14.70 `smtrat::options_detail` Namespace Reference

### Functions

- `void print_cmake_options ()`
- `void print_info ()`
- `void print_license ()`
- `void print_version ()`

#### 0.14.70.1 Function Documentation

**0.14.70.1.1 `print_cmake_options()`** `void smtrat::options_detail::print_cmake_options ( )`

**0.14.70.1.2 `print_info()`** `void smtrat::options_detail::print_info ( )`

**0.14.70.1.3 `print_license()`** `void smtrat::options_detail::print_license ( )`

**0.14.70.1.4 `print_version()`** `void smtrat::options_detail::print_version ( )`

## 0.14.71 `smtrat::parseformula` Namespace Reference

### Data Structures

- class [FormulaCollector](#)

## 0.14.72 `smtrat::parser` Namespace Reference

### Namespaces

- [arithmetic](#)
- [conversion](#)
- [core](#)
- [types](#)
- [uninterpreted](#)

### Data Structures

- struct [AttributeValueParser](#)
- struct [AttributeParser](#)
- struct [Skipper](#)
- struct [IdentifierParser](#)
- class [OutputWrapper](#)
- class [InstructionHandler](#)
- struct [RationalPolicies](#)

*Specialization of `qi::real_policies` for a Rational.*
- struct [NumeralParser](#)

*Parses numerals: (0 | [1-9] [0-9]\* )*
- struct [DecimalParser](#)

*Parses decimals: numeral . 0\*numeral*
- struct [HexadecimalParser](#)

*Parses hexadecimals: #x[0-9a-fA-F]+*
- struct [BinaryParser](#)

*Parses binaries: #b[01]+*
- struct [StringParser](#)

*Parses strings: ".+" with escape sequences \\ " and \\ \\\\"*
- struct [SimpleSymbolParser](#)

*Parses symbols: simple\_symbol | quoted\_symbol where.*
- struct [SymbolParser](#)
- struct [KeywordParser](#)

*Parses keywords: :simple\_symbol*
- class [SMTLIBParser](#)
- struct [ParserSettings](#)
- struct [LogicParser](#)

- struct [ErrorHandler](#)
- struct [QuantifierParser](#)
- struct [QEParser](#)
- struct [ScriptParser](#)
- struct [SpecConstantParser](#)
- struct [SExpressionParser](#)
- struct [SortParser](#)
- struct [QualifiedIdentifierParser](#)
- struct [SortedVariableParser](#)
- struct [TermParser](#)
- struct [AbstractTheory](#)
  - Base class for all theories.*
- struct [ArithmeticTheory](#)
  - Implements the theory of arithmetic, including LRA, LIA, NRA and NIA.*
- class [Attribute](#)
  - Represents an Attribute.*
- struct [BitvectorTheory](#)
  - Implements the theory of bitvectors.*
- struct [BooleanEncodingTheory](#)
  - Implements the theory of bitvectors.*
- struct [TheoryError](#)
- struct [Identifier](#)
- struct [SExpressionSequence](#)
- struct [CoreTheory](#)
  - Implements the core theory of the booleans.*
- struct [FunctionInstantiator](#)
- struct [IndexedFunctionInstantiator](#)
- struct [Instantiator](#)
- struct [UserFunctionInstantiator](#)
- struct [ParserState](#)
- struct [Theories](#)
  - The [Theories](#) class combines all implemented theories and provides a single interface to interact with all theories at once.*
- struct [FixedWidthConstant](#)
  - Represents a constant of a fixed width.*
- struct [UninterpretedTheory](#)
  - Implements the theory of equalities and uninterpreted functions.*

## TypeDefs

- `typedef boost::spirit::istream_iterator BaselteratorType`
- `typedef boost::spirit::line_pos_iterator< BaselteratorType > PositionIteratorType`
- `typedef PositionIteratorType Iterator`
- `template<typename T>`  
`using SExpression = boost::variant< T, boost::recursive_wrapper< SExpressionSequence< T > >>`

## Enumerations

- enum class [OptimizationType](#) { [Maximize](#) , [Minimize](#) }

## Functions

- std::ostream & `operator<<` (std::ostream &os, OptimizationType ot)
- template<typename T>  
void `registerParserSettings` (T &parser)
- std::ostream & `operator<<` (std::ostream &os, const Attribute &attr)
- void `at_most_k_helper` (size\_t k, const std::vector< FormulaT > &set, FormulasT &working, long long int prev\_idx, FormulasT &results)
- `FormulaT at_most_k` (size\_t k, const std::vector< FormulaT > &set)
- std::ostream & `operator<<` (std::ostream &os, const Identifier &identifier)
- template<typename T>  
std::ostream & `operator<<` (std::ostream &os, const SExpressionSequence< T > &ses)
- template<typename T>  
std::ostream & `operator<<` (std::ostream &os, const FixedWidthConstant< T > &fwc)

### 0.14.72.1 Typedef Documentation

**0.14.72.1.1 BaseliteratorType** `typedef boost::spirit::istream_iterator smtrat::parser::BaseIteratorType`

**0.14.72.1.2 Iterator** `typedef PositionIteratorType smtrat::parser::Iterator`

**0.14.72.1.3 PositionIteratorType** `typedef boost::spirit::line_pos_iterator<BaseIteratorType> smtrat::parser::PositionIteratorType`

**0.14.72.1.4 SExpression** `template<typename T>`  
`using smtrat::parser::SExpression = typedef boost::variant<T, boost::recursive_wrapper<SExpressionSequence<T>>>`

### 0.14.72.2 Enumeration Type Documentation

**0.14.72.2.1 OptimizationType** `enum smtrat::parser::OptimizationType [strong]`

#### Enumerator

|          |  |
|----------|--|
| Maximize |  |
| Minimize |  |

### 0.14.72.3 Function Documentation

**0.14.72.3.1 at\_most\_k()** `FormulaT smtrat::parser::at_most_k ( size_t k, const std::vector< FormulaT > & set )`

**0.14.72.3.2 at\_most\_k\_helper()** `void smtrat::parser::at_most_k_helper ( size_t k,`

```
 const std::vector< FormulaT > & set,
 FormulasT & working,
 long long int prev_idx,
 FormulasT & results)
```

**0.14.72.3.3 operator<<()** [1/5] std::ostream& smtrat::parser::operator<< ( std::ostream & os, const Attribute & attr ) [inline]

**0.14.72.3.4 operator<<()** [2/5] template<typename T> std::ostream& smtrat::parser::operator<< ( std::ostream & os, const FixedWidthConstant< T > & fwc ) [inline]

**0.14.72.3.5 operator<<()** [3/5] std::ostream& smtrat::parser::operator<< ( std::ostream & os, const Identifier & identifier ) [inline]

**0.14.72.3.6 operator<<()** [4/5] template<typename T> std::ostream& smtrat::parser::operator<< ( std::ostream & os, const SExpressionSequence< T > & ses ) [inline]

**0.14.72.3.7 operator<<()** [5/5] std::ostream& smtrat::parser::operator<< ( std::ostream & os, OptimizationType ot ) [inline]

**0.14.72.3.8 registerParserSettings()** template<typename T> void smtrat::parser::registerParserSettings ( T & parser )

## 0.14.73 smtrat::parser::arithmetic Namespace Reference

### TypeDefs

- using `OperatorType` = boost::variant< Poly::ConstructorOperation, carl::Relation >

### Functions

- `FormulaT makeConstraint (ArithmeticTheory &at, const Poly &lhs, const Poly &rhs, carl::Relation rel)`
- `bool isBooleanIdentity (const OperatorType &op, const std::vector< types::TermType > &arguments, TheoryError &errors)`

### 0.14.73.1 TypeDef Documentation

**0.14.73.1.1 OperatorType** using `smtrat::parser::arithmetic::OperatorType` = `typedef boost::variant<Poly::ConstructorOperation, carl::Relation>`

### 0.14.73.2 Function Documentation

**0.14.73.2.1 `isBooleanIdentity()`** `bool smtrat::parser::arithmetic::isBooleanIdentity (`  
 `const OperatorType & op,`  
 `const std::vector< types::TermType > & arguments,`  
 `TheoryError & errors ) [inline]`

**0.14.73.2.2 `makeConstraint()`** `FormulaT smtrat::parser::arithmetic::makeConstraint (`  
 `ArithmeticTheory & at,`  
 `const Poly & lhs,`  
 `const Poly & rhs,`  
 `carl::Relation rel ) [inline]`

## 0.14.74 smtrat::parser::conversion Namespace Reference

### Data Structures

- struct `Converter`
- struct `Converter< types::BVTerm >`
- struct `Converter< Poly >`
- struct `Converter< FormulaT >`
- struct `VariantConverter`

*Converts variants to some type using the `Converter` class.*
- struct `VariantVariantConverter`

*Converts variants to another variant type not using the `Converter` class.*
- struct `VectorVariantConverter`

*Converts a vector of variants to a vector of some type using the `Converter` class.*

## 0.14.75 smtrat::parser::core Namespace Reference

### Functions

- bool `convertTerm (const types::TermType &term, FormulaT &result)`
- bool `convertArguments (const std::vector< types::TermType > &arguments, std::vector< FormulaT > &result, TheoryError &errors)`

### 0.14.75.1 Function Documentation

**0.14.75.1.1 `convertArguments()`** `bool smtrat::parser::core::convertArguments (`  
 `const std::vector< types::TermType > & arguments,`  
 `std::vector< FormulaT > & result,`  
 `TheoryError & errors ) [inline]`

**0.14.75.1.2 `convertTerm()`** `bool smtrat::parser::core::convertTerm (`  
 `const types::TermType & term,`  
 `FormulaT & result ) [inline]`

## 0.14.76 smtrat::parser::types Namespace Reference

### Data Structures

- struct [CoreTheory](#)  
*Types of the core theory.*
- struct [ArithmeticTheory](#)  
*Types of the arithmetic theory.*
- struct [BitvectorTheory](#)  
*Types of the theory of bitvectors.*
- struct [UninterpretedTheory](#)  
*Types of the theory of equalities and uninterpreted functions.*

### Typedefs

- typedef carl::BVTerm [BVTerm](#)  
*Typedef for bitvector term.*
- typedef carl::BVVariable [BVVariable](#)
- typedef carl::BVConstraint [BVConstraint](#)  
*Typedef for bitvector constraint.*
- using [UTerm](#) = carl::UTerm
- typedef carl::mpl\_concatenate< [CoreTheory::ConstTypes](#), >::type [ConstTypes](#)  
*List of all types of constants.*
- typedef carl::mpl\_variant\_of< [ConstTypes](#) >::type [ConstType](#)  
*Variant type for all constants.*
- typedef carl::mpl\_concatenate< [CoreTheory::VariableTypes](#), >::type [VariableTypes](#)  
*List of all types of variables.*
- typedef carl::mpl\_variant\_of< [VariableTypes](#) >::type [VariableType](#)  
*Variant type for all variables.*
- typedef carl::mpl\_concatenate< [CoreTheory::ExpressionTypes](#), >::type [ExpressionTypes](#)  
*List of all types of expressions.*
- typedef carl::mpl\_variant\_of< [ExpressionTypes](#) >::type [ExpressionType](#)  
*Variant type for all expressions.*
- typedef carl::mpl\_concatenate< [CoreTheory::TermTypes](#), >::type [TermTypes](#)  
*List of all types of terms.*
- typedef carl::mpl\_variant\_of< [TermTypes](#) >::type [TermType](#)  
*Variant type for all terms.*
- typedef carl::mpl\_concatenate< [TermTypes](#), boost::mpl::vector< [SExpressionSequence< types::ConstType](#) >, std::string, bool, boost::spirit::unused\_type > >::type [AttributeTypes](#)  
*List of all types of attributes.*
- typedef carl::mpl\_variant\_of< [AttributeTypes](#) >::type [AttributeValue](#)  
*Variant type for all attributes.*

#### 0.14.76.1 Typedef Documentation

**0.14.76.1.1 AttributeTypes** typedef carl::mpl\_concatenate< [TermTypes](#), boost::mpl::vector< [SExpressionSequence< types::ConstType](#) >, std::string, bool, boost::spirit::unused\_type > >::type [smtrat::parser::types::AttributeTypes](#)  
List of all types of attributes.

**0.14.76.1.2 AttributeValue** typedef carl::mpl\_variant\_of<[AttributeTypes](#)>::type [smtrat::parser::types::AttributeValue](#)  
Variant type for all attributes.

**0.14.76.1.3 BVConstraint** `typedef carl::BVConstraint smrat::parser::types::BVConstraint`  
Typedef for bitvector constraint.

**0.14.76.1.4 BVTerm** `typedef carl::BVTerm smrat::parser::types::BVTerm`  
Typedef for bitvector term.

**0.14.76.1.5 BVVariable** `typedef carl::BVVariable smrat::parser::types::BVVariable`

**0.14.76.1.6 ConstType** `typedef carl::mpl_variant_of<ConstTypes>::type smrat::parser::types::ConstType`  
Variant type for all constants.

**0.14.76.1.7 ConstTypes** `typedef carl::mpl_concatenate< CoreTheory::ConstTypes, >::type smrat::parser::types::ConstTypes`  
List of all types of constants.

**0.14.76.1.8 ExpressionType** `typedef carl::mpl_variant_of<ExpressionTypes>::type smrat::parser::types::ExpressionType`  
Variant type for all expressions.

**0.14.76.1.9 ExpressionTypes** `typedef carl::mpl_concatenate< CoreTheory::ExpressionTypes, >::type smrat::parser::types::ExpressionTypes`  
List of all types of expressions.

**0.14.76.1.10 TermType** `typedef carl::mpl_variant_of<TermTypes>::type smrat::parser::types::TermType`  
Variant type for all terms.

**0.14.76.1.11 TermTypes** `typedef carl::mpl_concatenate< CoreTheory::TermTypes, >::type smrat::parser::types::TermTypes`  
List of all types of terms.

**0.14.76.1.12 UTerm** `using smrat::parser::types::UTerm = typedef carl::UTerm`

**0.14.76.1.13 VariableType** `typedef carl::mpl_variant_of<VariableTypes>::type smrat::parser::types::VariableType`  
Variant type for all variables.

**0.14.76.1.14 VariableTypes** `typedef carl::mpl_concatenate< CoreTheory::VariableTypes, >::type smrat::parser::types::VariableTypes`  
List of all types of variables.

## 0.14.77 smrat::parser::uninterpreted Namespace Reference

### Functions

- `bool convertTerm (const types::TermType &term, types::UTerm &result)`
- `bool convertArguments (const std::vector< types::TermType > &arguments, std::vector< types::UTerm > &result, TheoryError &errors)`

### 0.14.77.1 Function Documentation

```
0.14.77.1.1 convertArguments() bool smtrat::parser::uninterpreted::convertArguments (
 const std::vector< types::TermType > & arguments,
 std::vector< types::UTerm > & result,
 TheoryError & errors) [inline]
```

```
0.14.77.1.2 convertTerm() bool smtrat::parser::uninterpreted::convertTerm (
 const types::TermType & term,
 types::UTerm & result) [inline]
```

## 0.14.78 smtrat::preprocessor Namespace Reference

### 0.14.79 smtrat::qe Namespace Reference

#### Namespaces

- `cad`
- `fm`

#### TypeDefs

- using `QEQuery` = `std::vector< std::pair< QuantifierType, std::vector< carl::Variable > >>`

#### Enumerations

- enum class `QuantifierType` { `EXISTS` , `FORALL` }

#### Functions

- `std::ostream & operator<< (std::ostream &os, QuantifierType qt)`
- `std::ostream & operator<< (std::ostream &os, const QEQuery &q)`
- `std::vector< std::pair< QuantifierType, carl::Variable > > flattenQEQuery (const QEQuery &query)`
- `FormulaT eliminateQuantifiers (const FormulaT &qfree, const QEQuery &quantifiers)`

### 0.14.79.1 Typedef Documentation

```
0.14.79.1.1 QEQuery using smtrat::qe::QEQuery = typedef std::vector<std::pair<QuantifierType, std::vector<carl::Variable> >>
```

### 0.14.79.2 Enumeration Type Documentation

```
0.14.79.2.1 QuantifierType enum smtrat::qe::QuantifierType [strong]
```

#### Enumerator

|        |  |
|--------|--|
| EXISTS |  |
| FORALL |  |

### 0.14.79.3 Function Documentation

**0.14.79.3.1 `eliminateQuantifiers()`** `FormulaT smtrat::qe::eliminateQuantifiers (`  
 `const FormulaT & qfree,`  
 `const QEQuery & quantifiers ) [inline]`

**0.14.79.3.2 `flattenQEQuery()`** `std::vector<std::pair<QuantifierType, carl::Variable>> smtrat::qe::flattenQEQuery (`  
 `const QEQuery & query ) [inline]`

**0.14.79.3.3 `operator<<()` [1/2]** `std::ostream& smtrat::qe::operator<< (`  
 `std::ostream & os,`  
 `const QEQuery & q ) [inline]`

**0.14.79.3.4 `operator<<()` [2/2]** `std::ostream& smtrat::qe::operator<< (`  
 `std::ostream & os,`  
 `QuantifierType qt ) [inline]`

## 0.14.80 `smtrat::qe::cad` Namespace Reference

### Data Structures

- class [CAD](#)
- struct [CADSettings](#)
- class [CADElimination](#)
- class [Projection](#)

### Functions

- template<typename key , typename value >  
`std::multimap< value, key > flip_map (const std::map< key, value > &src)`
- template<typename S >  
`std::ostream & operator<< (std::ostream &os, const Projection< S > &p)`
- [FormulaT eliminateQuantifiers](#) (const [FormulaT](#) &qfree, const [QEQuery](#) &quantifiers)

### 0.14.80.1 Function Documentation

**0.14.80.1.1 `eliminateQuantifiers()`** `FormulaT smtrat::qe::cad::eliminateQuantifiers (`  
 `const FormulaT & qfree,`  
 `const QEQuery & quantifiers )`

**0.14.80.1.2 `flip_map()`** `template<typename key , typename value >`  
`std::multimap<value, key> smtrat::qe::cad::flip_map (`  
 `const std::map< key, value > & src )`

**0.14.80.1.3 `operator<<()`** `template<typename S >`  
`std::ostream& smtrat::qe::cad::operator<< (`  
 `std::ostream & os,`  
 `const Projection< S > & p )`

## 0.14.81 smtrat::qe::fm Namespace Reference

### Data Structures

- class [FourierMotzkinQE](#)

*Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints.*

### Functions

- [FormulaT eliminateQuantifiers](#) (const [FormulaT](#) &qfree, const [QEQuery](#) &quantifiers)

#### 0.14.81.1 Function Documentation

**0.14.81.1.1 [eliminateQuantifiers\(\)](#)** [FormulaT](#) smtrat::qe::fm::eliminateQuantifiers (

```
 const FormulaT & qfree,
 const QEQuery & quantifiers)
```

## 0.14.82 smtrat::resource Namespace Reference

### Data Structures

- class [Limiter](#)

### Functions

- void [setCPULimit](#) (std::size\_t)
- std::size\_t [getCPULimit](#) ()
- std::size\_t [usedCPU](#) ()
- void [setMemoryLimit](#) (std::size\_t)
- std::size\_t [getMemoryLimit](#) ()
- void [installSignalHandler](#) ()
- void [signalHandler](#) (int signal)

#### 0.14.82.1 Function Documentation

**0.14.82.1.1 [getCPULimit\(\)](#)** std::size\_t smtrat::resource::getCPULimit ( ) [inline]

**0.14.82.1.2 [getMemoryLimit\(\)](#)** std::size\_t smtrat::resource::getMemoryLimit ( ) [inline]

**0.14.82.1.3 [installSignalHandler\(\)](#)** void smtrat::resource::installSignalHandler ( ) [inline]

**0.14.82.1.4 [setCPULimit\(\)](#)** void smtrat::resource::setCPULimit (
 std::size\_t ) [inline]

**0.14.82.1.5 [setMemoryLimit\(\)](#)** void smtrat::resource::setMemoryLimit (
 std::size\_t ) [inline]

**0.14.82.1.6 signalHandler()** void smtrat::resource::signalHandler ( int *signal* ) [inline]

**0.14.82.1.7 usedCPU()** std::size\_t smtrat::resource::usedCPU ( ) [inline]

## 0.14.83 smtrat::sat Namespace Reference

### Namespaces

- [detail](#)

## 0.14.84 smtrat::sat::detail Namespace Reference

### Data Structures

- struct [ClauseChecker](#)

### Functions

- template<typename T>  
void [validateClause](#) (const T &*t*, bool *enabled*)
- template<typename T, typename VM, typename BCM>  
void [validateClause](#) (const T &*t*, const VM &*vm*, const BCM &*bcm*, bool *enabled*)

### 0.14.84.1 Function Documentation

**0.14.84.1.1 validateClause() [1/2]** template<typename T>  
void smtrat::sat::detail::validateClause ( const T & *t*,  
 bool *enabled* )

**0.14.84.1.2 validateClause() [2/2]** template<typename T, typename VM, typename BCM>  
void smtrat::sat::detail::validateClause ( const T & *t*,  
 const VM & *vm*,  
 const BCM & *bcm*,  
 bool *enabled* )

## 0.14.85 smtrat::settings Namespace Reference

### Data Structures

- struct [CoreSettings](#)
- struct [SolverSettings](#)
- struct [ModuleSettings](#)
- struct [Settings](#)

## 0.14.86 smtrat::statistics Namespace Reference

### Data Structures

- struct [StatisticsSettings](#)

## Functions

- template<typename T >  
void [registerStatisticsSettings](#) (T &parser)

### 0.14.86.1 Function Documentation

**0.14.86.1.1 registerStatisticsSettings()** template<typename T >  
void smtrat::statistics::registerStatisticsSettings (  
    T & parser )

## 0.14.87 smtrat::subtropical Namespace Reference

Implements data structures and encodings for the subtropical method.

### Data Structures

- struct [Moment](#)  
*Represents the normal vector component and the sign variable assigned to a variable in an original constraint.*
- struct [Vertex](#)  
*Represents a term of an original constraint and assigns him a rating variable if a weak separator is searched.*
- struct [Separator](#)  
*Represents the class of all original constraints with the same left hand side after a normalization.*
- class [Encoding](#)

### Enumerations

- enum class [SeparatorType](#) { **STRICT** = 0 , **WEAK** = 1 }
- enum class [Direction](#) { **BOTH** = 0 , **NEGATIVE** = 1 , **POSITIVE** = 2 }  
*Subdivides the relations into classes with the same linearization result.*

### Functions

- carl::BasicConstraint< [Poly](#) > [normalize](#) (const carl::BasicConstraint< [Poly](#) > &c)
- std::optional< [Direction](#) > [direction](#) (carl::Relation r)
- std::optional< [Direction](#) > [direction\\_for\\_equality](#) (const [Separator](#) &s)
- [FormulaT](#) [transform\\_to\\_equation](#) (const [FormulaT](#) &formula)  
*Requires a quantifier-free real arithmetic formula with no negations.*
- [FormulaT](#) [encode\\_as\\_formula](#) (const [FormulaT](#) &formula, [Encoding](#) &encoding, [SeparatorType](#) separator\_type)  
*Requires a quantifier-free real arithmetic formula with no negations.*

### 0.14.87.1 Detailed Description

Implements data structures and encodings for the subtropical method.

### 0.14.87.2 Enumeration Type Documentation

**Enumerator**

**0.14.87.2.1 Direction** enum `smtrat::subtropical::Direction` [strong]  
 Subdivides the relations into classes with the same linearization result.

**Enumerator**

|          |  |
|----------|--|
| BOTH     |  |
| NEGATIVE |  |
| POSITIVE |  |

**0.14.87.2.2 SeparatorType** enum `smtrat::subtropical::SeparatorType` [strong]

**Enumerator**

|        |  |
|--------|--|
| STRICT |  |
| WEAK   |  |

**0.14.87.3 Function Documentation**

**0.14.87.3.1 direction()** std::optional<`Direction`> `smtrat::subtropical::direction` (  
`carl::Relation r`) [inline]

**0.14.87.3.2 direction\_for\_equality()** std::optional<`Direction`> `smtrat::subtropical::direction_<=for_equality` (  
`const Separator & s`) [inline]

**0.14.87.3.3 encode\_as\_formula()** `FormulaT` `smtrat::subtropical::encode_as_formula` (  
`const FormulaT & formula,`  
`Encoding & encoding,`  
`SeparatorType separator_type`) [inline]

Requires a quantifier-free real arithmetic formula with no negations.

**Parameters**

|                      |                          |
|----------------------|--------------------------|
| <code>formula</code> | The formula to translate |
|----------------------|--------------------------|

**Returns**

linear formula

**0.14.87.3.4 normalize()** `carl::BasicConstraint<Poly>` `smtrat::subtropical::normalize` (  
`const carl::BasicConstraint< Poly > & c`) [inline]

**0.14.87.3.5 transform\_to\_equation()** `FormulaT` `smtrat::subtropical::transform_to_equation` (  
`const FormulaT & formula`) [inline]

Requires a quantifier-free real arithmetic formula with no negations.

#### Parameters

|                      |                          |
|----------------------|--------------------------|
| <code>formula</code> | The formula to transform |
|----------------------|--------------------------|

#### Returns

an equisatisfiable equation

## 0.14.88 smtrat::uf\_impl Namespace Reference

#### Data Structures

- class [uf\\_rank\\_and\\_class](#)  
*Wrapper class that contains the rank and parent class of some entry of the UF datastructure.*
- class [uf\\_rank\\_and\\_class< true >](#)  
*Compressed implementation of [uf\\_rank\\_and\\_class](#).*
- class [uf\\_data\\_wrapper](#)  
*Wrapper around data entry and [uf\\_rank\\_and\\_class](#); this class contains a value of type T and both rank and class of an entry in the UF datastructure.*
- class [uf\\_data\\_wrapper< void, compression >](#)  
*Special case of the [uf\\_data\\_wrapper](#) in case the data type is void.*
- struct [estimator](#)  
*Distance estimation for a range delimited by iterators; if the given iterators are random access, returns the result of std::distance in the estimate\_distance function.*
- struct [estimator< IteratorType, false >](#)

## 0.14.89 smtrat::unsatcore Namespace Reference

Contains strategy implementations for unsat core computations.

#### Data Structures

- class [UnsatCoreBackend](#)
- class [UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >](#)  
*Implements an easy strategy to obtain an unsatisfiable core.*

### 0.14.89.1 Detailed Description

Contains strategy implementations for unsat core computations.

## 0.14.90 smtrat::validation Namespace Reference

#### Data Structures

- class [ValidationCollector](#)
- class [ValidationPoint](#)
- struct [ValidationPrinter](#)
- struct [ValidationSettings](#)

#### Enumerations

- enum class [ValidationOutputFormat { SMTLIB }](#)

## Functions

- auto & `get` (const std::string &channel, const std::string &file, int line)
- template<ValidationOutputFormat SOF>  
std::ostream & `operator<<` (std::ostream &os, ValidationPrinter< SOF >)
- template<> std::ostream & `operator<<` (std::ostream &os, ValidationPrinter< ValidationOutputFormat::SMTLIB >)
- auto `validation_formulas_as_smtlib` ()
- void `validation_formulas_to_smtlib_file` (const std::string &filename)
- template<typename T >  
void `registerValidationSettings` (T &parser)

### 0.14.90.1 Enumeration Type Documentation

#### 0.14.90.1.1 ValidationOutputFormat enum smtrat::validation::ValidationOutputFormat [strong]

Enumerator

SMTLIB

### 0.14.90.2 Function Documentation

**0.14.90.2.1 get()** auto& smtrat::validation::get (  
const std::string & channel,  
const std::string & file,  
int line ) [inline]

**0.14.90.2.2 operator<<()** [1/2] template<ValidationOutputFormat SOF>  
std::ostream& smtrat::validation::operator<< (  
std::ostream & os,  
ValidationPrinter< SOF > )

**0.14.90.2.3 operator<<()** [2/2] template<>  
std::ostream& smtrat::validation::operator<< (  
std::ostream & os,  
ValidationPrinter< ValidationOutputFormat::SMTLIB > )

**0.14.90.2.4 registerValidationSettings()** template<typename T >  
void smtrat::validation::registerValidationSettings (  
T & parser )

**0.14.90.2.5 validation\_formulas\_as\_smtlib()** auto smtrat::validation::validation\_formulas\_as\_<→  
smtlib ( )

**0.14.90.2.6 validation\_formulas\_to\_smtlib\_file()** void smtrat::validation::validation\_formulas\_to\_<→  
smtlib\_file (   
const std::string & filename )

## 0.14.91 smtrat::vb Namespace Reference

### Data Structures

- class [Variable](#)  
*Class for a variable.*
- class [Bound](#)  
*Class for the bound of a variable.*
- class [VariableBounds](#)  
*Class to manage the bounds of a variable.*

### Functions

- template<typename Type >  
std::ostream & [operator<<](#) (std::ostream &\_os, const [VariableBounds< Type >](#) &\_vs)  
*Prints the contents of the given variable bounds manager to the given stream.*

#### 0.14.91.1 Function Documentation

**0.14.91.1.1 [operator<<\(\)](#)** template<typename Type >  
std::ostream& smtrat::vb::operator<< (  
    std::ostream & \_os,  
    const [VariableBounds< Type >](#) & \_vs ) [inline]

Prints the contents of the given variable bounds manager to the given stream.

#### Parameters

|                  |                                       |
|------------------|---------------------------------------|
| <code>_os</code> | The stream to print on.               |
| <code>_vs</code> | The variable bounds manager to print. |

#### Returns

The stream after printing on it.

## 0.14.92 smtrat::vs Namespace Reference

### Data Structures

- class [Condition](#)
- struct [unsignedTripleCmp](#)
- class [State](#)
- class [Substitution](#)
- struct [substitutionPointerEqual](#)
- struct [substitutionPointerHash](#)

### TypeDefs

- typedef std::set< carl::PointerSet< [Condition](#) > > [ConditionSetSet](#)
- typedef std::set< [ConditionSetSet](#) > [ConditionSetSetSet](#)
- typedef std::list< const [Condition](#) \* > [ConditionList](#)
- typedef std::vector< [ConditionList](#) > [DisjunctionOfConditionConjunctions](#)
- typedef std::pair< size\_t, std::pair< size\_t, size\_t > > [UnsignedTriple](#)
- typedef std::pair< [UnsignedTriple](#), [vs::State](#) \* > [ValStatePair](#)
- typedef std::map< [UnsignedTriple](#), [vs::State](#) \*, [unsignedTripleCmp](#) > [ValuationMap](#)
- typedef std::vector< [smtrat::ConstraintT](#) > [ConstraintVector](#)

- `a vector of constraints`
- `typedef std::vector< ConstraintVector > DisjunctionOfConstraintConjunctions`  
`a vector of vectors of constraints`
- `template<typename T >`  
`using SubstitutionFastPointerMap = std::unordered_map< const smrat::Poly *, T, substitutionPointerHash, substitutionPointerEqual >`

## Functions

- `std::ostream & operator<< (std::ostream &_out, const Condition &_condition)`
- `void simplify (DisjunctionOfConstraintConjunctions &)`  
*Simplifies a disjunction of conjunctions of constraints by deleting consistent constraint and inconsistent conjunctions of constraints.*
- `void simplify (DisjunctionOfConstraintConjunctions &_toSimplify, Variables &_conflictingVars, const smrat::EvalDoubleIntervalMap &_solutionSpace)`
- `bool splitProducts (DisjunctionOfConstraintConjunctions &, bool=false)`  
*Splits all constraints in the given disjunction of conjunctions of constraints having a non-trivial factorization into a set of constraints which compare the factors instead.*
- `bool splitProducts (const ConstraintVector &, DisjunctionOfConstraintConjunctions &, bool=false)`  
*Splits all constraints in the given conjunction of constraints having a non-trivial factorization into a set of constraints which compare the factors instead.*
- `DisjunctionOfConstraintConjunctions splitProducts (const smrat::ConstraintT &, bool=false)`  
*Splits the given constraint into a set of constraints which compare the factors of the factorization of the constraints considered polynomial.*
- `void splitSosDecompositions (DisjunctionOfConstraintConjunctions &_toSimplify)`
- `DisjunctionOfConstraintConjunctions getSignCombinations (const smrat::ConstraintT &)`  
*For a given constraint  $f_1 * \dots * f_n \sim 0$  this method computes all combinations of constraints  $f_1 \sim 1 \dots$*
- `void getOddBitStrings (std::size_t _length, std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > &_strings)`
- `void getEvenBitStrings (std::size_t _length, std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > > &_strings)`
- `void print (DisjunctionOfConstraintConjunctions &_substitutionResults)`  
*Prints the given disjunction of conjunction of constraints.*
- `bool substitute (const smrat::ConstraintT &_cons, const Substitution &_subs, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper, Variables &_conflictingVariables, const smrat::EvalDoubleIntervalMap &↔_solutionSpace)`
- `bool substituteNormal (const smrat::ConstraintT &_cons, const Substitution &_subs, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper, Variables &_conflictingVariables, const smrat::EvalDoubleIntervalMap &↔_solutionSpace)`
- `bool substituteNormalSqrtEq (const smrat::Poly &_radicand, const smrat::Poly &_q, const smrat::Poly &_r, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper)`  
*Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.*
- `bool substituteNormalSqrtNeq (const smrat::Poly &_radicand, const smrat::Poly &_q, const smrat::Poly &↔_r, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper)`  
*Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.*
- `bool substituteNormalSqrtLess (const smrat::Poly &_radicand, const smrat::Poly &_q, const smrat::Poly &_r, const smrat::Poly &_s, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper)`  
*Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.*
- `bool substituteNormalSqrtLew (const smrat::Poly &_radicand, const smrat::Poly &_q, const smrat::Poly &↔_r, const smrat::Poly &_s, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper)`  
*Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.*
- `bool substitutePlusEps (const smrat::ConstraintT &_cons, const Substitution &_subs, DisjunctionOfConstraintConjunctions &_result, bool _accordingPaper, Variables &_conflictingVariables, const smrat::EvalDoubleIntervalMap &↔_solutionSpace)`

- bool `substituteEpsGradients` (const `smrat::ConstraintT` &`_cons`, const `Substitution` &`_subs`, const `Relation` &`_relation`, `DisjunctionOfConstraintConjunctions` &`_result`, bool `_accordingPaper`, `Variables` &`_conflictingVariables`, const `smrat::EvalDoubleIntervalMap` &`_solutionSpace`)
- void `substituteInf` (const `smrat::ConstraintT` &`_cons`, const `Substitution` &`_subs`, `DisjunctionOfConstraintConjunctions` &`_result`, `Variables` &`_conflictingVariables`, const `smrat::EvalDoubleIntervalMap` &`_solutionSpace`)
- void `substituteInfLessGreater` (const `smrat::ConstraintT` &`_cons`, const `Substitution` &`_subs`, `DisjunctionOfConstraintConjunctions` &`_result`)  
*Applies the given substitution to the given constraint, where the substitution is of the form [x -> +/-infinity] with x as the variable and c and b polynomials in the real theory excluding x.*
- void `substituteTrivialCase` (const `smrat::ConstraintT` &`_cons`, const `Substitution` &`_subs`, `DisjunctionOfConstraintConjunctions` &`_result`)  
*Deals with the case, that the left hand side of the constraint to substitute is a trivial polynomial in the variable to substitute.*
- void `substituteNotTrivialCase` (const `smrat::ConstraintT` &, const `Substitution` &, `DisjunctionOfConstraintConjunctions` &)  
*Deals with the case, that the left hand side of the constraint to substitute is not a trivial polynomial in the variable to substitute.*
- template<class `combineType`>  
 bool `combine` (const `std::vector< std::vector< std::vector< combineType > > >` &`_toCombine`, `std::vector< std::vector< combineType > >` &`_combination`)  
*Combines vectors.*
- void `simplify` (`DisjunctionOfConstraintConjunctions` &, `carl::Variables` &, const `smrat::EvalDoubleIntervalMap` &)  
*Simplifies a disjunction of conjunctions of constraints by deleting consistent constraint and inconsistent conjunctions of constraints.*
- void `getOddBitStrings` (size\_t `_length`, `std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > >` &`_strings`)
- void `getEvenBitStrings` (size\_t `_length`, `std::vector< std::bitset< MAX_PRODUCT_SPLIT_NUMBER > >` &`_strings`)
- bool `substitute` (const `smrat::ConstraintT` &, const `Substitution` &, `DisjunctionOfConstraintConjunctions` &, bool `_accordingPaper`, `carl::Variables` &, const `smrat::EvalDoubleIntervalMap` &)  
*Applies a substitution to a constraint and stores the results in the given vector.*
- bool `substituteNormal` (const `smrat::ConstraintT` &`_cons`, const `Substitution` &`_subs`, `DisjunctionOfConstraintConjunctions` &`_result`, bool `_accordingPaper`, `carl::Variables` &`_conflictingVariables`, const `smrat::EvalDoubleIntervalMap` &`_solutionSpace`)  
*Applies a substitution of a variable to a term, which is not minus infinity nor a to an square root expression plus an infinitesimal.*
- bool `substitutePlusEps` (const `smrat::ConstraintT` &`_cons`, const `Substitution` &`_subs`, `DisjunctionOfConstraintConjunctions` &`_result`, bool `_accordingPaper`, `carl::Variables` &`_conflictingVariables`, const `smrat::EvalDoubleIntervalMap` &`_solutionSpace`)  
*Applies the given substitution to the given constraint, where the substitution is of the form [x -> t+epsilon] with x as the variable and c and b polynomials in the real theory excluding x.*
- bool `substituteEpsGradients` (const `smrat::ConstraintT` &`_cons`, const `Substitution` &`_subs`, const `carl::Relation` &`_relation`, `DisjunctionOfConstraintConjunctions` &, bool `_accordingPaper`, `carl::Variables` &`_conflictingVariables`, const `smrat::EvalDoubleIntervalMap` &`_solutionSpace`)  
*Sub-method of `substituteEps`, where one of the gradients in the point represented by the substitution must be negative if the given relation is less or positive if the given relation is greater.*
- void `substituteInf` (const `smrat::ConstraintT` &`_cons`, const `Substitution` &`_subs`, `DisjunctionOfConstraintConjunctions` &`_result`, `carl::Variables` &`_conflictingVariables`, const `smrat::EvalDoubleIntervalMap` &`_solutionSpace`)  
*Applies the given substitution to the given constraint, where the substitution is of the form [x -> -infinity] with x as the variable and c and b polynomials in the real theory excluding x.*
- `std::ostream` & `operator<<` (`std::ostream` &`os`, const `Substitution` &`s`)  
*Prints the given substitution on the given output stream.*

### 0.14.92.1 Typedef Documentation

**0.14.92.1.1 ConditionList** `typedef std::list< const Condition* > smtrat::vs::ConditionList`

**0.14.92.1.2 ConditionSetSet** `typedef std::set< carl::PointerSet<Condition> > smtrat::vs::ConditionSetSet`

**0.14.92.1.3 ConditionSetSetSet** `typedef std::set< ConditionSetSet > smtrat::vs::ConditionSetSetSet`

**0.14.92.1.4 ConstraintVector** `typedef std::vector<smtrat::ConstraintT> smtrat::vs::ConstraintVector`  
a vector of constraints

**0.14.92.1.5 DisjunctionOfConditionConjunctions** `typedef std::vector< ConditionList > smtrat::vs::DisjunctionOfConditionConjunctions`

**0.14.92.1.6 DisjunctionOfConstraintConjunctions** `typedef std::vector<ConstraintVector> smtrat::vs::DisjunctionOfConstraintConjunctions`  
a vector of vectors of constraints

```
0.14.92.1.7 SubstitutionFastPointerMap template<typename T >
using smtrat::vs::SubstitutionFastPointerMap = typedef std::unordered_map<const smtrat::Poly*, T, substitutionPointerHash, substitutionPointerEqual>
```

**0.14.92.1.8 UnsignedTriple** `typedef std::pair<size_t, std::pair<size_t, size_t> > smtrat::vs::UnsignedTriple`

**0.14.92.1.9 ValStatePair** `typedef std::pair<UnsignedTriple, vs::State*> smtrat::vs::ValStatePair`

**0.14.92.1.10 ValuationMap** `typedef std::map<UnsignedTriple, vs::State*, unsignedTripleCmp> smtrat::vs::ValuationMap`

### 0.14.92.2 Function Documentation

```
0.14.92.2.1 combine() template<class combineType >
bool smtrat::vs::combine (
 const std::vector< std::vector< std::vector< combineType > > > & _toCombine,
 std::vector< std::vector< combineType > > & _combination)
```

Combines vectors.

#### Parameters

|                           |                            |
|---------------------------|----------------------------|
| <code>_toCombine</code>   | The vectors to combine.    |
| <code>_combination</code> | The resulting combination. |

**Returns**

false, if the upper limit in the number of combinations resulting by this method is exceeded. Note, that this hinders a combinatorial blow up. true, otherwise.

**0.14.92.2.2 getEvenBitStrings() [1/2]** void smtrat::vs::getEvenBitStrings ( size\_t \_length,  
std::vector< std::bitset< MAX\_PRODUCT\_SPLIT\_NUMBER > > & \_strings )

**Parameters**

|          |                                                                            |
|----------|----------------------------------------------------------------------------|
| _length  | The maximal length of the bit strings with even parity to compute.         |
| _strings | All bit strings of length less or equal the given length with even parity. |

**0.14.92.2.3 getEvenBitStrings() [2/2]** void smtrat::vs::getEvenBitStrings ( std::size\_t \_length,  
std::vector< std::bitset< MAX\_PRODUCT\_SPLIT\_NUMBER > > & \_strings )

**0.14.92.2.4 getOddBitStrings() [1/2]** void smtrat::vs::getOddBitStrings ( size\_t \_length,  
std::vector< std::bitset< MAX\_PRODUCT\_SPLIT\_NUMBER > > & \_strings )

**Parameters**

|          |                                                                           |
|----------|---------------------------------------------------------------------------|
| _length  | The maximal length of the bit strings with odd parity to compute.         |
| _strings | All bit strings of length less or equal the given length with odd parity. |

**0.14.92.2.5 getOddBitStrings() [2/2]** void smtrat::vs::getOddBitStrings ( std::size\_t \_length,  
std::vector< std::bitset< MAX\_PRODUCT\_SPLIT\_NUMBER > > & \_strings )

**0.14.92.2.6 getSignCombinations()** *DisjunctionOfConstraintConjunctions* smtrat::vs::getSign← Combinations ( const smtrat::ConstraintT & )

For a given constraint  $f_1 \sim \dots \sim f_n \sim 0$  this method computes all combinations of constraints  $f_1 \sim_1 0 \dots f_n \sim_n 0$  such that

$f_1 \sim_1 0 \text{ and } \dots \text{ and } f_n \sim_n 0 \quad \text{iff} \quad f_1 \sim \dots \sim f_n \sim 0$

holds.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| _constraint | A pointer to the constraint to split this way. |
|-------------|------------------------------------------------|

**Returns**

The resulting combinations.

**0.14.92.2.7 operator<<()** [1/2] `std::ostream& smtrat::vs::operator<< ( std::ostream & _out, const Condition & _condition )`

**0.14.92.2.8 operator<<()** [2/2] `std::ostream & smtrat::vs::operator<< ( std::ostream & os, const Substitution & s )`

Prints the given substitution on the given output stream.

#### Parameters

|                            |                                       |
|----------------------------|---------------------------------------|
| <code>_out</code>          | The output stream, on which to write. |
| <code>_substitution</code> | The substitution to print.            |

#### Returns

The output stream after printing the substitution on it.

**0.14.92.2.9 print()** `void smtrat::vs::print ( DisjunctionOfConstraintConjunctions & _substitutionResults )`

Prints the given disjunction of conjunction of constraints.

#### Parameters

|                                   |                                                         |
|-----------------------------------|---------------------------------------------------------|
| <code>_substitutionResults</code> | The disjunction of conjunction of constraints to print. |
|-----------------------------------|---------------------------------------------------------|

**0.14.92.2.10 simplify()** [1/3] `void smtrat::vs::simplify ( DisjunctionOfConstraintConjunctions & )`

Simplifies a disjunction of conjunctions of constraints by deleting consistent constraint and inconsistent conjunctions of constraints.

If a conjunction of only consistent constraints exists, the simplified disjunction contains one empty conjunction.

#### Parameters

|                          |                                              |
|--------------------------|----------------------------------------------|
| <code>_toSimplify</code> | The disjunction of conjunctions to simplify. |
|--------------------------|----------------------------------------------|

**0.14.92.2.11 simplify()** [2/3] `void smtrat::vs::simplify ( DisjunctionOfConstraintConjunctions & , carl::Variables & , const smtrat::EvalDoubleIntervalMap & )`

Simplifies a disjunction of conjunctions of constraints by deleting consistent constraint and inconsistent conjunctions of constraints.

If a conjunction of only consistent constraints exists, the simplified disjunction contains one empty conjunction.

#### Parameters

|                               |                                              |
|-------------------------------|----------------------------------------------|
| <code>_toSimplify</code>      | The disjunction of conjunctions to simplify. |
| <code>_conflictingVars</code> |                                              |
| <code>_solutionSpace</code>   |                                              |

---

**0.14.92.2.12** `simplify()` [3/3] `void smtrat::vs::simplify (`  
`DisjunctionOfConstraintConjunctions & _toSimplify,`  
`Variables & _conflictingVars,`  
`const smtrat::EvalDoubleIntervalMap & _solutionSpace )`

**0.14.92.2.13** `splitProducts()` [1/3] `bool smtrat::vs::splitProducts (`  
`const ConstraintVector & ,`  
`DisjunctionOfConstraintConjunctions & ,`  
`bool = false )`

Splits all constraints in the given conjunction of constraints having a non-trivial factorization into a set of constraints which compare the factors instead.

#### Parameters

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| <code>_toSimplify</code> | The conjunction of the constraints to split.                                   |
| <code>_result</code>     | The result, being a disjunction of conjunctions of constraints.                |
| <code>_onlyNeq</code>    | A flag indicating that only constraints with the relation symbol != are split. |

#### Returns

false, if the upper limit in the number of combinations resulting by this method is exceeded. Note, that this hinders a combinatorial blow up. true, otherwise.

**0.14.92.2.14** `splitProducts()` [2/3] `DisjunctionOfConstraintConjunctions smtrat::vs::splitProducts (`  
`const smtrat::ConstraintT & ,`  
`bool = false )`

Splits the given constraint into a set of constraints which compare the factors of the factorization of the constraints considered polynomial.

#### Parameters

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| <code>_constraint</code> | A pointer to the constraint to split.                                          |
| <code>_onlyNeq</code>    | A flag indicating that only constraints with the relation symbol != are split. |

#### Returns

The resulting disjunction of conjunctions of constraints, which is semantically equivalent to the given constraint.

**0.14.92.2.15** `splitProducts()` [3/3] `bool smtrat::vs::splitProducts (`  
`DisjunctionOfConstraintConjunctions & ,`  
`bool = false )`

Splits all constraints in the given disjunction of conjunctions of constraints having a non-trivial factorization into a set of constraints which compare the factors instead.

#### Parameters

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| <code>_toSimplify</code> | The disjunction of conjunctions of the constraints to split.                   |
| <code>_onlyNeq</code>    | A flag indicating that only constraints with the relation symbol != are split. |

**Returns**

false, if the upper limit in the number of combinations resulting by this method is exceeded. Note, that this hinders a combinatorial blow up. true, otherwise.

**0.14.92.2.16 `splitSosDecompositions()`** void smrat::vs::splitSosDecompositions (   
`DisjunctionOfConstraintConjunctions` & `_toSimplify` )

**0.14.92.2.17 `substitute()` [1/2]** bool smrat::vs::substitute (   
 const `smrat::ConstraintT` & ,   
 const `Substitution` & ,   
`DisjunctionOfConstraintConjunctions` & ,   
 bool `_accordingPaper`,   
`carl::Variables` & ,   
 const `smrat::EvalDoubleIntervalMap` & )

Applies a substitution to a constraint and stores the results in the given vector.

**Parameters**

|                      |                                                                 |
|----------------------|-----------------------------------------------------------------|
| <code>_cons</code>   | The constraint to substitute in.                                |
| <code>_subs</code>   | The substitution to apply.                                      |
| <code>_result</code> | The vector, in which to store the results of this substitution. |

**Returns**

false, if the upper limit in the number of combinations in the result of the substitution is exceeded. Note, that this hinders a combinatorial blow up. true, otherwise.

**0.14.92.2.18 `substitute()` [2/2]** bool smrat::vs::substitute (   
 const `smrat::ConstraintT` & `_cons`,   
 const `Substitution` & `_subs`,   
`DisjunctionOfConstraintConjunctions` & `_result`,   
 bool `_accordingPaper`,   
`Variables` & `_conflictingVariables`,   
 const `smrat::EvalDoubleIntervalMap` & `_solutionSpace` )

**0.14.92.2.19 `substituteEpsGradients()` [1/2]** bool smrat::vs::substituteEpsGradients (   
 const `smrat::ConstraintT` & `_cons`,   
 const `Substitution` & `_subs`,   
 const `carl::Relation` `_relation`,   
`DisjunctionOfConstraintConjunctions` & ,   
 bool `_accordingPaper`,   
`carl::Variables` & `_conflictingVariables`,   
 const `smrat::EvalDoubleIntervalMap` & `_solutionSpace` )

Sub-method of `substituteEps`, where one of the gradients in the point represented by the substitution must be negative if the given relation is less or positive if the given relation is greater.

**Parameters**

|                    |                                  |
|--------------------|----------------------------------|
| <code>_cons</code> | The constraint to substitute in. |
| <code>_subs</code> | The substitution to apply.       |

**Parameters**

|                                    |                                                                                                                                                                                                                                                                                                                     |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_relation</code>             | The relation symbol, deciding whether the substitution result must be negative or positive.                                                                                                                                                                                                                         |
| <code>_result</code>               | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.                                                                                                                                                                                    |
| <code>_accordingPaper</code>       | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |
| <code>_conflictingVariables</code> | If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container.                                                                                                                                                                                         |
| <code>_solutionSpace</code>        | The solution space in form of double intervals of the variables occurring in the given constraint.                                                                                                                                                                                                                  |

```
0.14.92.2.20 substituteEpsGradients() [2/2] bool smtrat::vs::substituteEpsGradients (
 const smtrat::ConstraintT & _cons,
 const Substitution & _subs,
 const Relation _relation,
 DisjunctionOfConstraintConjunctions & _result,
 bool _accordingPaper,
 Variables & _conflictingVariables,
 const smtrat::EvalDoubleIntervalMap & _solutionSpace)
```

```
0.14.92.2.21 substituteInf() [1/2] void smtrat::vs::substituteInf (
 const smtrat::ConstraintT & _cons,
 const Substitution & _subs,
 DisjunctionOfConstraintConjunctions & _result,
 carl::Variables & _conflictingVariables,
 const smtrat::EvalDoubleIntervalMap & _solutionSpace)
```

Applies the given substitution to the given constraint, where the substitution is of the form  $[x \rightarrow -\infty]$  with  $x$  as the variable and  $c$  and  $b$  polynomials in the real theory excluding  $x$ .

The constraint is of the form " $f(x) \rho 0$ " with  $\rho$  element of  $\{=, !=, <, >, \leq, \geq\}$  and  $k$  as the maximum degree of  $x$  in  $f$ .

**Parameters**

|                                    |                                                                                                                                  |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <code>_cons</code>                 | The constraint to substitute in.                                                                                                 |
| <code>_subs</code>                 | The substitution to apply.                                                                                                       |
| <code>_result</code>               | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |
| <code>_conflictingVariables</code> | If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container.      |
| <code>_solutionSpace</code>        | The solution space in form of double intervals of the variables occurring in the given constraint.                               |

```
0.14.92.2.22 substituteInf() [2/2] void smtrat::vs::substituteInf (
 const smtrat::ConstraintT & _cons,
 const Substitution & _subs,
 DisjunctionOfConstraintConjunctions & _result,
 Variables & _conflictingVariables,
```

```
const smtrat::EvalDoubleIntervalMap & _solutionSpace)
```

**0.14.92.2.23 substituteInfLessGreater()** void smtrat::vs::substituteInfLessGreater (

```
const smtrat::ConstraintT & _cons,
const Substitution & _subs,
DisjunctionOfConstraintConjunctions & _result)
```

Applies the given substitution to the given constraint, where the substitution is of the form  $[x \rightarrow +/-\infty]$  with x as the variable and c and b polynomials in the real theory excluding x.

The constraint is of the form " $a*x^2+bx+c \rho 0$ ", where  $\rho$  is less or greater.

#### Parameters

|                |                                                                                                                                  |
|----------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>_cons</i>   | The constraint to substitute in.                                                                                                 |
| <i>_subs</i>   | The substitution to apply.                                                                                                       |
| <i>_result</i> | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |

**0.14.92.2.24 substituteNormal() [1/2]** bool smtrat::vs::substituteNormal (

```
const smtrat::ConstraintT & _cons,
const Substitution & _subs,
DisjunctionOfConstraintConjunctions & _result,
bool _accordingPaper,
carl::Variables & _conflictingVariables,
const smtrat::EvalDoubleIntervalMap & _solutionSpace)
```

Applies a substitution of a variable to a term, which is not minus infinity nor a to an square root expression plus an infinitesimal.

#### Parameters

|                              |                                                                                                                                                                                                                                                                                                                     |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>_cons</i>                 | The constraint to substitute in.                                                                                                                                                                                                                                                                                    |
| <i>_subs</i>                 | The substitution to apply.                                                                                                                                                                                                                                                                                          |
| <i>_result</i>               | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.                                                                                                                                                                                    |
| <i>_accordingPaper</i>       | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |
| <i>_conflictingVariables</i> | If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container.                                                                                                                                                                                         |
| <i>_solutionSpace</i>        | The solution space in form of double intervals of the variables occurring in the given constraint.                                                                                                                                                                                                                  |

**0.14.92.2.25 substituteNormal() [2/2]** bool smtrat::vs::substituteNormal (

```
const smtrat::ConstraintT & _cons,
const Substitution & _subs,
DisjunctionOfConstraintConjunctions & _result,
bool _accordingPaper,
Variables & _conflictingVariables,
const smtrat::EvalDoubleIntervalMap & _solutionSpace)
```

---

```
0.14.92.2.26 substituteNormalSqrtEq() bool smtrat::vs::substituteNormalSqrtEq (
 const smtrat::Poly & _radicand,
 const smtrat::Poly & _q,
 const smtrat::Poly & _r,
 DisjunctionOfConstraintConjunctions & _result,
 bool _accordingPaper)
```

Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.  
The relation symbol of the constraint to substitute is "=".

$$(_q + _r * \sqrt{(_radicand)})$$

The term then looks like: ----- \_s

#### Parameters

|                        |                                                                                                                                                                                                                                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>_radicand</i>       | The radicand of the square root.                                                                                                                                                                                                                                                                                    |
| <i>_q</i>              | The summand not containing the square root.                                                                                                                                                                                                                                                                         |
| <i>_r</i>              | The coefficient of the radicand.                                                                                                                                                                                                                                                                                    |
| <i>_result</i>         | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.                                                                                                                                                                                    |
| <i>_accordingPaper</i> | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |

---

```
0.14.92.2.27 substituteNormalSqrtLeq() bool smtrat::vs::substituteNormalSqrtLeq (
 const smtrat::Poly & _radicand,
 const smtrat::Poly & _q,
 const smtrat::Poly & _r,
 const smtrat::Poly & _s,
 DisjunctionOfConstraintConjunctions & _result,
 bool _accordingPaper)
```

Sub-method of substituteNormalSqrt, where applying the substitution led to a term containing a square root.  
The relation symbol of the constraint to substitute is less or equal.

$$(_q + _r * \sqrt{(_radicand)})$$

The term then looks like: ----- \_s

#### Parameters

|                        |                                                                                                                                                                                                                                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>_radicand</i>       | The radicand of the square root.                                                                                                                                                                                                                                                                                    |
| <i>_q</i>              | The summand not containing the square root.                                                                                                                                                                                                                                                                         |
| <i>_r</i>              | The coefficient of the radicand.                                                                                                                                                                                                                                                                                    |
| <i>_s</i>              | The denominator of the expression containing the square root.                                                                                                                                                                                                                                                       |
| <i>_result</i>         | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.                                                                                                                                                                                    |
| <i>_accordingPaper</i> | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |

---

```
0.14.92.2.28 substituteNormalSqrtLess() bool smtrat::vs::substituteNormalSqrtLess (
 const smtrat::Poly & _radicand,
```

```

const smtrat::Poly & _q,
const smtrat::Poly & _r,
const smtrat::Poly & _s,
DisjunctionOfConstraintConjunctions & _result,
bool _accordingPaper)

```

Sub-method of `substituteNormalSqrt`, where applying the substitution led to a term containing a square root.  
The relation symbol of the constraint to substitute is less.

$$(\_q + \_r * \sqrt{(\_radicand)})$$

The term then looks like: -----  $\_s$

#### Parameters

|                              |                                                                                                                                                                                                                                                                                                                     |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_radicand</code>       | The radicand of the square root.                                                                                                                                                                                                                                                                                    |
| <code>_q</code>              | The summand not containing the square root.                                                                                                                                                                                                                                                                         |
| <code>_r</code>              | The coefficient of the radicand.                                                                                                                                                                                                                                                                                    |
| <code>_s</code>              | The denominator of the expression containing the square root.                                                                                                                                                                                                                                                       |
| <code>_result</code>         | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.                                                                                                                                                                                    |
| <code>_accordingPaper</code> | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |

**0.14.92.2.29 `substituteNormalSqrtNeq()`** `bool smtrat::vs::substituteNormalSqrtNeq (`

```

const smtrat::Poly & _radicand,
const smtrat::Poly & _q,
const smtrat::Poly & _r,
DisjunctionOfConstraintConjunctions & _result,
bool _accordingPaper)

```

Sub-method of `substituteNormalSqrt`, where applying the substitution led to a term containing a square root.  
The relation symbol of the constraint to substitute is " $!=$ ".

$$(\_q + \_r * \sqrt{(\_radicand)})$$

The term then looks like: -----  $\_s$

#### Parameters

|                              |                                                                                                                                                                                                                                                                                                                     |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_radicand</code>       | The radicand of the square root.                                                                                                                                                                                                                                                                                    |
| <code>_q</code>              | The summand not containing the square root.                                                                                                                                                                                                                                                                         |
| <code>_r</code>              | The coefficient of the radicand.                                                                                                                                                                                                                                                                                    |
| <code>_result</code>         | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.                                                                                                                                                                                    |
| <code>_accordingPaper</code> | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |

**0.14.92.2.30 `substituteNotTrivialCase()`** `void smtrat::vs::substituteNotTrivialCase (`

```

const smtrat::ConstraintT & ,
const Substitution & ,
DisjunctionOfConstraintConjunctions &)

```

Deals with the case, that the left hand side of the constraint to substitute is not a trivial polynomial in the variable to substitute.

The constraints left hand side then should looks like:  $ax^2+bx+c$

#### Parameters

|                      |                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <code>_cons</code>   | The constraint to substitute in.                                                                                                 |
| <code>_subs</code>   | The substitution to apply.                                                                                                       |
| <code>_result</code> | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |

```
0.14.92.2.31 substitutePlusEps() [1/2] bool smtrat::vs::substitutePlusEps (
 const smtrat::ConstraintT & _cons,
 const Substitution & _subs,
 DisjunctionOfConstraintConjunctions & _result,
 bool _accordingPaper,
 carl::Variables & _conflictingVariables,
 const smtrat::EvalDoubleIntervalMap & _solutionSpace)
```

Applies the given substitution to the given constraint, where the substitution is of the form  $[x \rightarrow t+\epsilon]$  with  $x$  as the variable and  $c$  and  $b$  polynomials in the real theory excluding  $x$ .

The constraint is of the form " $f(x) \rho 0$ " with  $\rho$  element of  $\{=, !=, <, >, \leq, \geq\}$  and  $k$  as the maximum degree of  $x$  in  $f$ .

#### Parameters

|                                    |                                                                                                                                                                                                                                                                                                                     |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_cons</code>                 | The constraint to substitute in.                                                                                                                                                                                                                                                                                    |
| <code>_subs</code>                 | The substitution to apply.                                                                                                                                                                                                                                                                                          |
| <code>_result</code>               | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints.                                                                                                                                                                                    |
| <code>_accordingPaper</code>       | A flag that indicates whether to apply the virtual substitution rules according to the paper "Quantifier elimination for real algebra - the quadratic case and beyond." by Volker Weispfenning (true) or in an adapted way which omits a higher degree in the result by splitting the result in more cases (false). |
| <code>_conflictingVariables</code> | If a conflict with the given solution space occurs, the variables being part of this conflict are stored in this container.                                                                                                                                                                                         |
| <code>_solutionSpace</code>        | The solution space in form of double intervals of the variables occurring in the given constraint.                                                                                                                                                                                                                  |

```
0.14.92.2.32 substitutePlusEps() [2/2] bool smtrat::vs::substitutePlusEps (
 const smtrat::ConstraintT & _cons,
 const Substitution & _subs,
 DisjunctionOfConstraintConjunctions & _result,
 bool _accordingPaper,
 Variables & _conflictingVariables,
 const smtrat::EvalDoubleIntervalMap & _solutionSpace)
```

```
0.14.92.2.33 substituteTrivialCase() void smtrat::vs::substituteTrivialCase (
 const smtrat::ConstraintT & _cons,
 const Substitution & _subs,
 DisjunctionOfConstraintConjunctions & _result)
```

Deals with the case, that the left hand side of the constraint to substitute is a trivial polynomial in the variable to substitute.

The constraints left hand side then should look like:  $ax^2+bx+c$

#### Parameters

|                      |                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <code>_cons</code>   | The constraint to substitute in.                                                                                                 |
| <code>_subs</code>   | The substitution to apply.                                                                                                       |
| <code>_result</code> | The vector, in which to store the results of this substitution. It is semantically a disjunction of conjunctions of constraints. |

## 0.15 Data Structure Documentation

### 0.15.1 smrat::AbstractModuleFactory Struct Reference

```
#include <StrategyGraph.h>
```

#### Public Member Functions

- virtual `~AbstractModuleFactory ()=default`
- virtual `Module * create (const ModuleInput *_formula, Conditionals &_conditionals, Manager *const _manager)=0`
- virtual std::string `moduleName () const =0`

#### 0.15.1.1 Constructor & Destructor Documentation

**0.15.1.1.1 `~AbstractModuleFactory()`** virtual smrat::AbstractModuleFactory::~AbstractModuleFactory ( ) [virtual], [default]

#### 0.15.1.2 Member Function Documentation

**0.15.1.2.1 `create()`** virtual Module\* smrat::AbstractModuleFactory::create ( const ModuleInput \* `_formula`, Conditionals & `_conditionals`, Manager \*const `_manager` ) [pure virtual]

Implemented in `smrat::ModuleFactory< Module >`.

**0.15.1.2.2 `moduleName()`** virtual std::string smrat::AbstractModuleFactory::moduleName ( ) const [pure virtual]

Implemented in `smrat::ModuleFactory< Module >`.

### 0.15.2 smrat::parser::AbstractTheory Struct Reference

Base class for all theories.

```
#include <AbstractTheory.h>
```

#### Public Member Functions

- `AbstractTheory (ParserState *state)`
- virtual `~AbstractTheory ()`

- virtual bool `declareVariable` (const std::string &, const carl::Sort &, types::VariableType &, TheoryError &errors)  
*Declare a new variable with the given name and the given sort.*
- virtual bool `resolveSymbol` (const Identifier &, types::TermType &, TheoryError &errors)  
*Resolve a symbol that was not declared within the ParserState.*
- virtual bool `handleITE` (const FormulaT &, const types::TermType &, const types::TermType &, types::TermType &, TheoryError &errors)  
*Resolve an if-then-else operator.*
- virtual bool `handleDistinct` (const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)  
*Resolve a distinct operator.*
- template<typename T , typename Builder >  
    `FormulaT expandDistinct` (const std::vector< T > &values, const Builder &neqBuilder)
- virtual bool `instantiate` (const types::VariableType &, const types::TermType &, types::TermType &, TheoryError &errors)  
*Instantiate a variable within a term.*
- virtual bool `refreshVariable` (const types::VariableType &, types::VariableType &, TheoryError &errors)
- virtual bool `functionCall` (const Identifier &, const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)  
*Resolve another unknown function call.*

## Static Public Member Functions

- static void `addSimpleSorts` (qi::symbols< char, carl::Sort > &)  
*Initialize the global symbol table for simple sorts.*
- static void `addConstants` (qi::symbols< char, types::ConstType > &)  
*Initialize the global symbol table for constants.*

## Data Fields

- ParserState \* state

### 0.15.2.1 Detailed Description

Base class for all theories.

A theory represents one or multiple SMT-LIB theories and takes care of converting smtlib constructs into our data-structures.

A theory has two kinds of functions:

- Initializer are static functions that initialize global datastructures. These are for example the set of constants.
- Handlers are member functions that implement a certain SMT-LIB functionality. Handlers always return a boolean that indicates if the theory could complete the request. Oftentimes, a theory will return false because the request should be handled by another theory. Handlers are used, whenever the parser can not easily decide which theory to use and thus tries to call every theory. The result of a handler is always a term that is returned through a reference argument.

A theory may override any of the following methods.

### 0.15.2.2 Constructor & Destructor Documentation

#### 0.15.2.2.1 AbstractTheory()

```
smtrat::parser::AbstractTheory::AbstractTheory (
 ParserState * state) [inline]
```

#### 0.15.2.2.2 ~AbstractTheory()

```
virtual smtrat::parser::AbstractTheory::~AbstractTheory () [inline], [virtual]
```

### 0.15.2.3 Member Function Documentation

**0.15.2.3.1 addConstants()** static void smtrat::parser::AbstractTheory::addConstants (qi::symbols< char, types::ConstType > & ) [inline], [static]

Initialize the global symbol table for constants.

**0.15.2.3.2 addSimpleSorts()** static void smtrat::parser::AbstractTheory::addSimpleSorts (qi::symbols< char, carl::Sort > & ) [inline], [static]

Initialize the global symbol table for simple sorts.

**0.15.2.3.3 declareVariable()** virtual bool smtrat::parser::AbstractTheory::declareVariable (const std::string & , const carl::Sort & , types::VariableType & , TheoryError & errors ) [inline], [virtual]

Declare a new variable with the given name and the given sort.

Reimplemented in [smtrat::parser::UninterpretedTheory](#), [smtrat::parser::CoreTheory](#), [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

**0.15.2.3.4 expandDistinct()** template<typename T , typename Builder > FormulaT smtrat::parser::AbstractTheory::expandDistinct (const std::vector< T > & values, const Builder & negBuilder ) [inline]

**0.15.2.3.5 functionCall()** virtual bool smtrat::parser::AbstractTheory::functionCall (const Identifier & , const std::vector< types::TermType > & , types::TermType & , TheoryError & errors ) [inline], [virtual]

Resolve another unknown function call.

Reimplemented in [smtrat::parser::UninterpretedTheory](#), [smtrat::parser::CoreTheory](#), [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

**0.15.2.3.6 handleDistinct()** virtual bool smtrat::parser::AbstractTheory::handleDistinct (const std::vector< types::TermType > & , types::TermType & , TheoryError & errors ) [inline], [virtual]

Resolve a distinct operator.

Reimplemented in [smtrat::parser::CoreTheory](#), [smtrat::parser::BitvectorTheory](#), [smtrat::parser::ArithmeticTheory](#), and [smtrat::parser::UninterpretedTheory](#).

**0.15.2.3.7 handleITE()** virtual bool smtrat::parser::AbstractTheory::handleITE (const FormulaT & , const types::TermType & , const types::TermType & , types::TermType & , TheoryError & errors ) [inline], [virtual]

Resolve an if-then-else operator.

Reimplemented in [smtrat::parser::UninterpretedTheory](#), [smtrat::parser::CoreTheory](#), [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

**0.15.2.3.8 `instantiate()`** `virtual bool smtrat::parser::AbstractTheory::instantiate (`  
    `const types::VariableType &,`  
    `const types::TermType &,`  
    `types::TermType &,`  
    `TheoryError & errors ) [inline], [virtual]`

Instantiate a variable within a term.

Reimplemented in [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

**0.15.2.3.9 `refreshVariable()`** `virtual bool smtrat::parser::AbstractTheory::refreshVariable (`  
    `const types::VariableType &,`  
    `types::VariableType &,`  
    `TheoryError & errors ) [inline], [virtual]`

Reimplemented in [smtrat::parser::BitvectorTheory](#).

**0.15.2.3.10 `resolveSymbol()`** `virtual bool smtrat::parser::AbstractTheory::resolveSymbol (`  
    `const Identifier &,`  
    `types::TermType &,`  
    `TheoryError & errors ) [inline], [virtual]`

Resolve a symbol that was not declared within the [ParserState](#).

This might be a symbol that actually uses indices, for example bitvector constants.

Reimplemented in [smtrat::parser::BitvectorTheory](#).

#### 0.15.2.4 Field Documentation

**0.15.2.4.1 `state`** `ParserState* smtrat::parser::AbstractTheory::state`

### 0.15.3 [smtrat::cad::sample\\_compare::absvalue Struct Reference](#)

```
#include <SampleComparator.h>
```

### 0.15.4 [smtrat::analyzer::AnalysisSettings Struct Reference](#)

```
#include <settings.h>
```

#### Data Fields

- `bool enabled = false`
- `std::string analyze_projections = "none"`

#### 0.15.4.1 Field Documentation

**0.15.4.1.1 `analyze_projections`** `std::string smtrat::analyzer::AnalysisSettings::analyze_projections = "none"`

**0.15.4.1.2 `enabled`** `bool smtrat::analyzer::AnalysisSettings::enabled = false`

## 0.15.5 smrat::analyzer::AnalyzerStatistics Struct Reference

```
#include <statistics.h>
```

### Public Member Functions

- template<typename T >  
void **add** (const std::string &key, const T &value)

#### 0.15.5.1 Member Function Documentation

**0.15.5.1.1 add()** template<typename T >  
void smrat::analyzer::AnalyzerStatistics::add (  
 const std::string & key,  
 const T & value ) [inline]

## 0.15.6 smrat::AnnotatedBVTerm Class Reference

```
#include <IntBlastModule.h>
```

### Public Member Functions

- **AnnotatedBVTerm ()**
- **AnnotatedBVTerm** (const **BVAnnotation** &\_type, const carl::BVTerm &\_term)
- **AnnotatedBVTerm** (std::size\_t \_width, bool \_signed, **Integer** \_offset=0)
- **AnnotatedBVTerm** (const **BVAnnotation** &\_type)
- const **BVAnnotation** & **type** () const
- const carl::BVTerm & **term** () const
- const carl::BVTerm & **operator()** () const

### Friends

- std::ostream & **operator<<** (std::ostream &\_out, const **AnnotatedBVTerm** &\_term)

#### 0.15.6.1 Constructor & Destructor Documentation

**0.15.6.1.1 AnnotatedBVTerm() [1/4]** smrat::AnnotatedBVTerm::AnnotatedBVTerm ( ) [inline]

**0.15.6.1.2 AnnotatedBVTerm() [2/4]** smrat::AnnotatedBVTerm::AnnotatedBVTerm (   
 const **BVAnnotation** & \_type,  
 const carl::BVTerm & \_term ) [inline]

**0.15.6.1.3 AnnotatedBVTerm() [3/4]** smrat::AnnotatedBVTerm::AnnotatedBVTerm (   
 std::size\_t \_width,  
 bool \_signed,  
 **Integer** \_offset = 0 ) [inline]

**0.15.6.1.4 AnnotatedBVTerm() [4/4]** smrat::AnnotatedBVTerm::AnnotatedBVTerm (   
 const **BVAnnotation** & \_type ) [inline]

### 0.15.6.2 Member Function Documentation

**0.15.6.2.1 operator()** const carl::BVTerm& smtrat::AnnotatedBVTerm::operator() () const [inline]

**0.15.6.2.2 term()** const carl::BVTerm& smtrat::AnnotatedBVTerm::term () const [inline]

**0.15.6.2.3 type()** const BVAnnotation& smtrat::AnnotatedBVTerm::type () const [inline]

### 0.15.6.3 Friends And Related Function Documentation

**0.15.6.3.1 operator<<** std::ostream& operator<< ( std::ostream & \_out, const AnnotatedBVTerm & \_term ) [friend]

## 0.15.7 smtrat::cadcells::representation::approximation::ApxCriteria Class Reference

```
#include <criteria.h>
```

### Static Public Member Functions

- static void **inform** (const Polynomial &p, std::size\_t root\_index)
- static bool **cell** (const std::vector< Atom > &constraints)
- static bool **level** (std::size\_t lvl)
- static bool **poly** (datastructures::Projections &proj, const IR &ir)
- static bool **poly\_pair** (datastructures::Projections &proj, const IR &ir\_l, const IR &ir\_u)
- static bool **side** (datastructures::Projections &proj, const IR &ir, datastructures::RootMap::const\_iterator start, datastructures::RootMap::const\_iterator end)

### 0.15.7.1 Member Function Documentation

**0.15.7.1.1 cell()** static bool smtrat::cadcells::representation::approximation::ApxCriteria::cell ( const std::vector< Atom > & constraints ) [inline], [static]

**0.15.7.1.2 inform()** static void smtrat::cadcells::representation::approximation::ApxCriteria::inform ( const Polynomial & p, std::size\_t root\_index ) [inline], [static]

**0.15.7.1.3 level()** static bool smtrat::cadcells::representation::approximation::ApxCriteria::level ( std::size\_t lvl ) [inline], [static]

```
0.15.7.1.4 poly() static bool smtrat::cadcells::representation::approximation::ApxCriteria<→
::poly (
 datastructures::Projections & proj,
 const IR & ir) [inline], [static]
```

```
0.15.7.1.5 poly_pair() static bool smtrat::cadcells::representation::approximation::ApxCriteria<→
::poly_pair (
 datastructures::Projections & proj,
 const IR & ir_l,
 const IR & ir_u) [inline], [static]
```

```
0.15.7.1.6 side() static bool smtrat::cadcells::representation::approximation::ApxCriteria<→
::side (
 datastructures::Projections & proj,
 const IR & ir,
 datastructures::RootMap::const_iterator start,
 datastructures::RootMap::const_iterator end) [inline], [static]
```

## 0.15.8 smtrat::cadcells::representation::approximation::ApxSettings Struct Reference

```
#include <ApproximationSettings.h>
```

### Data Fields

- const std::size\_t `taylor_deg = settings_module().get("apx_taylor_deg", 1)`
- const std::size\_t `maximize_n_iter = settings_module().get("apx_maximize_iter", 5)`
- const std::size\_t `n_sb_iterations = settings_module().get("apx_sb_iter", 2)`
- const double `root_ratio_lower = settings_module().get("apx_root_ratio_l", 0.75)`
- const double `root_ratio_upper = settings_module().get("apx_root_ratio_u", 0.875)`
- const std::size\_t `crit_max_considered = settings_module().get("apx_max_considered", 20)`
- const std::size\_t `crit_max_apx = settings_module().get("apx_max_apx", 50)`
- const std::size\_t `crit_max_constraint_involved = settings_module().get("apx_max_involved", 5)`
- const std::size\_t `crit_max_apx_per_poly = settings_module().get("apx_max_app", 5)`
- const std::size\_t `crit_degree_threshold = settings_module().get("apx_deg_threshold", 5)`
- const bool `crit_level_enabled = settings_module().get("apx_level_enabled", false)`
- const bool `crit_considered_count_enabled = settings_module().get("apx_considered_enabled", false)`
- const bool `crit_apx_count_enabled = settings_module().get("apx_count_enabled", true)`
- const bool `crit_single_degree_enabled = settings_module().get("apx_single_degree_enabled", true)`
- const bool `crit_pair_degree_enabled = settings_module().get("apx_pair_degree_enabled", false)`
- const bool `crit_poly_apx_count_enabled = settings_module().get("apx_poly_count_enabled", false)`
- const bool `crit_involved_count_enabled = settings_module().get("apx_involved_count_enabled", false)`
- const bool `crit_side_degree_enabled = settings_module().get("apx_side_degree_enabled", false)`

### Static Public Attributes

- static constexpr ApxPoly `bound = ApxPoly::SIMPLE`
- static constexpr ApxPoly `between = ApxPoly::SIMPLE`
- static constexpr ApxRoot `root = ApxRoot::SIMPLE_REPRESENTATION`

### 0.15.8.1 Field Documentation

**0.15.8.1.1 `between`** `constexpr ApxPoly smtrat::cadcells::representation::approximation::ApxSettings::between = ApxPoly::SIMPLE [static], [constexpr]`

**0.15.8.1.2 `bound`** `constexpr ApxPoly smtrat::cadcells::representation::approximation::ApxSettings::bound = ApxPoly::SIMPLE [static], [constexpr]`

**0.15.8.1.3 `crit_apx_count_enabled`** `const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_apx_count_enabled = settings_module().get("apx_count_enabled", true)`

**0.15.8.1.4 `crit_considered_count_enabled`** `const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_considered_count_enabled = settings_module().get("apx_considered_enabled", false)`

**0.15.8.1.5 `crit_degree_threshold`** `const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::crit_degree_threshold = settings_module().get("apx_deg_threshold", 5)`

**0.15.8.1.6 `crit_involved_count_enabled`** `const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_involved_count_enabled = settings_module().get("apx_involved_count_enabled", false)`

**0.15.8.1.7 `crit_level_enabled`** `const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_level_enabled = settings_module().get("apx_level_enabled", false)`

**0.15.8.1.8 `crit_max_apx`** `const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::crit_max_apx = settings_module().get("apx_max_apx", 50)`

**0.15.8.1.9 `crit_max_apx_per_poly`** `const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::crit_max_apx_per_poly = settings_module().get("apx_max_app", 5)`

**0.15.8.1.10 `crit_max_considered`** `const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::crit_max_considered = settings_module().get("apx_max_considered", 20)`

**0.15.8.1.11 `crit_max_constraint_involved`** `const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::crit_max_constraint_involved = settings_module().get("apx_max_involved", 5)`

**0.15.8.1.12 `crit_pair_degree_enabled`** `const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_pair_degree_enabled = settings_module().get("apx_pair_degree_enabled", false)`

**0.15.8.1.13 `crit_poly_apx_count_enabled`** `const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_poly_apx_count_enabled = settings_module().get("apx_poly_count_enabled", false)`

```
0.15.8.1.14 crit_side_degree_enabled const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_side_degree_enabled = settings_module().get("apx_side_degree_enabled", false)
```

```
0.15.8.1.15 crit_single_degree_enabled const bool smtrat::cadcells::representation::approximation::ApxSettings::crit_single_degree_enabled = settings_module().get("apx_single_degree_enabled", true)
```

```
0.15.8.1.16 maximize_n_iter const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::maximize_n_iter = settings_module().get("apx_maximize_iter", 5)
```

```
0.15.8.1.17 n_sb_iterations const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::n_sb_iterations = settings_module().get("apx_sb_iter", 2)
```

```
0.15.8.1.18 root constexpr ApxRoot smtrat::cadcells::representation::approximation::ApxSettings::root = ApxRoot::SIMPLE REPRESENTATION [static], [constexpr]
```

```
0.15.8.1.19 root_ratio_lower const double smtrat::cadcells::representation::approximation::ApxSettings::root_ratio_lower = settings_module().get("apx_root_ratio_l", 0.75)
```

```
0.15.8.1.20 root_ratio_upper const double smtrat::cadcells::representation::approximation::ApxSettings::root_ratio_upper = settings_module().get("apx_root_ratio_u", 0.875)
```

```
0.15.8.1.21 taylor_deg const std::size_t smtrat::cadcells::representation::approximation::ApxSettings::taylor_deg = settings_module().get("apx_taylor_deg", 1)
```

## 0.15.9 benchmax::slurm::ArchiveProperties Struct Reference

All properties needed to archive log files.  
#include <SlurmUtilities.h>

### Data Fields

- std::string **filename\_archive**  
*Filename of the archive.*
- std::string **tmp\_dir**  
*Temporary directory to look for output files.*

### 0.15.9.1 Detailed Description

All properties needed to archive log files.

### 0.15.9.2 Field Documentation

```
0.15.9.2.1 filename_archive std::string benchmax::slurm::ArchiveProperties::filename_archive
Filename of the archive.
```

**0.15.9.2.2 tmp\_dir** std::string benchmax::slurm::ArchiveProperties::tmp\_dir  
Temporary directory to look for output files.

## 0.15.10 smtrat::parser::ArithmeticTheory Struct Reference

Implements the theory of arithmetic, including LRA, LIA, NRA and NIA.

```
#include <Arithmetic.h>
```

### Public Member Functions

- **ArithmeticTheory (ParserState \*state)**  
*Declare a new variable with the given name and the given sort.*
- **bool declareVariable (const std::string &name, const carl::Sort &sort, types::VariableType &result, TheoryError &errors)**  
*Resolve an if-then-else operator.*
- **bool handleITE (const FormulaT &ifterm, const types::TermType &thenterm, const types::TermType &elseterm, types::TermType &result, TheoryError &errors)**  
*Resolve a distinct operator.*
- **bool instantiate (const types::VariableType &var, const types::TermType &replacement, types::TermType &result, TheoryError &errors)**  
*Instantiate a variable within a term.*
- **bool functionCall (const Identifier &identifier, const std::vector< types::TermType > &arguments, types::TermType &result, types::TermType &result, TheoryError &errors)**  
*Resolve another unknown function call.*
- **virtual bool resolveSymbol (const Identifier &, types::TermType &, TheoryError &errors)**  
*Resolve a symbol that was not declared within the ParserState.*
- **template<typename T, typename Builder> FormulaT expandDistinct (const std::vector< T > &values, const Builder &neqBuilder)**
- **virtual bool refreshVariable (const types::VariableType &, types::VariableType &, TheoryError &errors)**

### Static Public Member Functions

- **static void addSimpleSorts (qi::symbols< char, carl::Sort > &sorts)**  
*Initialize the global symbol table for constants.*
- **static void addConstants (qi::symbols< char, types::ConstType > &)**

### Data Fields

- **std::map< std::string, arithmetic::OperatorType > ops**
- **std::map< carl::Variable, std::tuple< FormulaT, Poly, Poly > > mITEs**
- **ParserState \* state**

### 0.15.10.1 Detailed Description

Implements the theory of arithmetic, including LRA, LIA, NRA and NIA.

### 0.15.10.2 Constructor & Destructor Documentation

**0.15.10.2.1 ArithmeticTheory()** smtrat::parser::ArithmeticTheory::ArithmeticTheory (  
    ParserState \* state )

### 0.15.10.3 Member Function Documentation

**0.15.10.3.1 `addConstants()`** static void smtrat::parser::AbstractTheory::addConstants (qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]  
Initialize the global symbol table for constants.

**0.15.10.3.2 `addSimpleSorts()`** void smtrat::parser::ArithmeticTheory::addSimpleSorts (qi::symbols< char, carl::Sort > & sorts ) [static]

**0.15.10.3.3 `declareVariable()`** bool smtrat::parser::ArithmeticTheory::declareVariable (const std::string &, const carl::Sort &, types::VariableType &, TheoryError & errors ) [virtual]

Declare a new variable with the given name and the given sort.

Reimplemented from [smtrat::parser::AbstractTheory](#).

**0.15.10.3.4 `expandDistinct()`** template<typename T , typename Builder > FormulaT smtrat::parser::AbstractTheory::expandDistinct (const std::vector< T > & values, const Builder & negBuilder ) [inline], [inherited]

**0.15.10.3.5 `functionCall()`** bool smtrat::parser::ArithmeticTheory::functionCall (const Identifier &, const std::vector< types::TermType > &, types::TermType &, TheoryError & errors ) [virtual]

Resolve another unknown function call.

Reimplemented from [smtrat::parser::AbstractTheory](#).

**0.15.10.3.6 `handleDistinct()`** bool smtrat::parser::ArithmeticTheory::handleDistinct (const std::vector< types::TermType > &, types::TermType &, TheoryError & errors ) [virtual]

Resolve a distinct operator.

Reimplemented from [smtrat::parser::AbstractTheory](#).

**0.15.10.3.7 `handleITE()`** bool smtrat::parser::ArithmeticTheory::handleITE (const FormulaT &, const types::TermType &, const types::TermType &, types::TermType &, TheoryError & errors ) [virtual]

Resolve an if-then-else operator.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.10.3.8 instantiate() bool smtrat::parser::ArithmeticTheory::instantiate (
 const types::VariableType & ,
 const types::TermType & ,
 types::TermType & ,
 TheoryError & errors) [virtual]
```

Instantiate a variable within a term.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.10.3.9 refreshVariable() virtual bool smtrat::parser::AbstractTheory::refreshVariable (
 const types::VariableType & ,
 types::VariableType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Reimplemented in [smtrat::parser::BitvectorTheory](#).

```
0.15.10.3.10 resolveSymbol() virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
 const Identifier & ,
 types::TermType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Resolve a symbol that was not declared within the [ParserState](#).

This might be a symbol that actually uses indices, for example bitvector constants.

Reimplemented in [smtrat::parser::BitvectorTheory](#).

## 0.15.10.4 Field Documentation

```
0.15.10.4.1 mITES std::map<carl::Variable, std::tuple<FormulaT, Poly, Poly>> smtrat::parser<-
 ::ArithmeticTheory::mITES
```

```
0.15.10.4.2 ops std::map<std::string, arithmetic::OperatorType> smtrat::parser::Arithmetic<-
 Theory::ops
```

```
0.15.10.4.3 state ParserState* smtrat::parser::AbstractTheory::state [inherited]
```

## 0.15.11 smtrat::parser::types::ArithmeticTheory Struct Reference

Types of the arithmetic theory.

```
#include <TheoryTypes.h>
```

### Public Types

- `typedef mpl::vector< Rational, FixedWidthConstant< Integer > > ConstTypes`
- `typedef mpl::vector< carl::Variable > VariableTypes`
- `typedef mpl::vector< carl::Variable, Rational, FixedWidthConstant< Integer >, Poly, carl::MultivariateRoot< Poly > > ExpressionTypes`
- `typedef mpl::vector< carl::Variable, Rational, FixedWidthConstant< Integer >, Poly, carl::MultivariateRoot< Poly > > TermTypes`
- `typedef carl::mpl_variant_of< TermTypes >::type TermType`

### 0.15.11.1 Detailed Description

Types of the arithmetic theory.

### 0.15.11.2 Member Typedef Documentation

**0.15.11.2.1 ConstTypes** `typedef mpl::vector<Rational, FixedWidthConstant<Integer> > smtrat::parser::types::ConstTypes`

**0.15.11.2.2 ExpressionTypes** `typedef mpl::vector<carl::Variable, Rational, FixedWidthConstant<Integer>, Poly, carl::MultivariateRoot<Poly> > smtrat::parser::types::ArithmeticTheory::ExpressionTypes`

**0.15.11.2.3 TermType** `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::ArithmeticTheory::TermType`

**0.15.11.2.4 TermTypes** `typedef mpl::vector<carl::Variable, Rational, FixedWidthConstant<Integer>, Poly, carl::MultivariateRoot<Poly> > smtrat::parser::types::ArithmeticTheory::TermTypes`

**0.15.11.2.5 VariableTypes** `typedef mpl::vector<carl::Variable> smtrat::parser::types::ArithmeticTheory::VariableTypes`

## 0.15.12 smtrat::execution::Assertion Struct Reference

```
#include <ExecutionState.h>
```

### Data Fields

- `FormulaT formula`

### 0.15.12.1 Field Documentation

**0.15.12.1.1 formula** `FormulaT smtrat::execution::Assertion::formula`

## 0.15.13 smtrat::cad::preprocessor::AssignmentCollector Class Reference

```
#include <CADPreprocessor.h>
```

### Public Types

- using `CollectionResult = std::variant< bool, ConstraintT >`  
*Result of an assignment collection.*

### Public Member Functions

- `AssignmentCollector (Model &model)`
- `const auto & reasons () const`
- `auto & reasons ()`
- `const auto & constraints () const`
- `auto & constraints ()`
- `CollectionResult collect (std::map< ConstraintT, ConstraintT > &constraints)`

### 0.15.13.1 Member Typedef Documentation

**0.15.13.1.1 CollectionResult** using `smtrat::cad::preprocessor::AssignmentCollector::CollectionResult`  
= `std::variant<bool, ConstraintT>`  
Result of an assignment collection.  
true: new assignments were found false: no new assignments were found constraint: found direct conflict

## 0.15.13.2 Constructor & Destructor Documentation

**0.15.13.2.1 AssignmentCollector()** `smtrat::cad::preprocessor::AssignmentCollector::AssignmentCollector(`  
`Model & model ) [inline]`

## 0.15.13.3 Member Function Documentation

**0.15.13.3.1 collect()** `AssignmentCollector::CollectionResult smtrat::cad::preprocessor::AssignmentCollector::collect(`  
`std::map< ConstraintT, ConstraintT > & constraints )`

**0.15.13.3.2 constraints() [1/2]** `auto& smtrat::cad::preprocessor::AssignmentCollector::constraints( ) [inline]`

**0.15.13.3.3 constraints() [2/2]** `const auto& smtrat::cad::preprocessor::AssignmentCollector::constraints( ) const [inline]`

**0.15.13.3.4 reasons() [1/2]** `auto& smtrat::cad::preprocessor::AssignmentCollector::reasons( ) [inline]`

**0.15.13.3.5 reasons() [2/2]** `const auto& smtrat::cad::preprocessor::AssignmentCollector::reasons( ) const [inline]`

## 0.15.14 smtrat::mcsat::arithmetic::AssignmentFinder Struct Reference

```
#include <AssignmentFinder.h>
```

### Public Member Functions

- `std::optional< AssignmentOrConflict > operator()(const mcsat::Bookkeeping &data, carl::Variable var) const`
- `bool active(const mcsat::Bookkeeping &data, const FormulaT &f) const`

## 0.15.14.1 Member Function Documentation

**0.15.14.1.1 active()** `bool smtrat::mcsat::arithmetic::AssignmentFinder::active(`  
`const mcsat::Bookkeeping & data,`  
`const FormulaT & f ) const`

```
0.15.14.1.2 operator() std::optional< AssignmentOrConflict > smtrat::mcsat::arithmetic::↔
AssignmentFinder::operator() (
 const mcsat::Bookkeeping & data,
 carl::Variable var) const
```

## 0.15.15 smtrat::mcsat::smtaf::AssignmentFinder< Settings > Struct Template Reference

```
#include <AssignmentFinder.h>
```

### Public Member Functions

- std::optional< AssignmentOrConflict > **operator()** (const mcsat::Bookkeeping &data, carl::Variable var) const
- bool **active** (const mcsat::Bookkeeping &, const FormulaT &) const

#### 0.15.15.1 Member Function Documentation

```
0.15.15.1.1 active() template<class Settings >
bool smtrat::mcsat::smtaf::AssignmentFinder< Settings >::active (
 const mcsat::Bookkeeping & ,
 const FormulaT &) const [inline]
```

```
0.15.15.1.2 operator() template<class Settings >
std::optional<AssignmentOrConflict> smtrat::mcsat::smtaf::AssignmentFinder< Settings >↔
::operator() (
 const mcsat::Bookkeeping & data,
 carl::Variable var) const [inline]
```

## 0.15.16 smtrat::mcsat::arithmetic::AssignmentFinder\_ctx Class Reference

```
#include <AssignmentFinder_ctx.h>
```

### Public Member Functions

- **AssignmentFinder\_ctx** (const std::vector< carl::Variable > &var\_order, carl::Variable var, const Model &model)
- bool **addConstraint** (const FormulaT &f1)
- bool **addMVBound** (const FormulaT &f1)
- **Covering computeCover ()**
- **AssignmentOrConflict findAssignment ()**

#### 0.15.16.1 Constructor & Destructor Documentation

```
0.15.16.1.1 AssignmentFinder_ctx() smtrat::mcsat::arithmetic::AssignmentFinder_ctx::Assignment↔
Finder_ctx (
 const std::vector< carl::Variable > & var_order,
 carl::Variable var,
 const Model & model) [inline]
```

#### 0.15.16.2 Member Function Documentation

```
0.15.16.2.1 addConstraint() bool smtrat::mcsat::arithmetic::AssignmentFinder_ctx::addConstraint
(
 const FormulaT & f1) [inline]
```

```
0.15.16.2.2 addMVBound() bool smtrat::mcsat::arithmetic::AssignmentFinder_ctx::addMVBound (
 const FormulaT & f1) [inline]
```

```
0.15.16.2.3 computeCover() Covering smtrat::mcsat::arithmetic::AssignmentFinder_ctx::compute←
Cover () [inline]
```

```
0.15.16.2.4 findAssignment() AssignmentOrConflict smtrat::mcsat::arithmetic::AssignmentFinder←
_ctx::findAssignment () [inline]
```

## 0.15.17 smtrat::mcsat::arithmetic::AssignmentFinder\_detail Class Reference

```
#include <AssignmentFinder_arithmetic.h>
```

### Public Member Functions

- [AssignmentFinder\\_detail](#) (carl::Variable var, const Model &model)
- bool [addConstraint](#) (const FormulaT &f)
- bool [addMVBound](#) (const FormulaT &f)
- [Covering computeCover](#) ()
- [AssignmentOrConflict findAssignment](#) ()

### 0.15.17.1 Constructor & Destructor Documentation

```
0.15.17.1.1 AssignmentFinder_detail() smtrat::mcsat::arithmetic::AssignmentFinder_detail::←
AssignmentFinder_detail (
 carl::Variable var,
 const Model & model) [inline]
```

### 0.15.17.2 Member Function Documentation

```
0.15.17.2.1 addConstraint() bool smtrat::mcsat::arithmetic::AssignmentFinder_detail::add←
Constraint (
 const FormulaT & f) [inline]
```

```
0.15.17.2.2 addMVBound() bool smtrat::mcsat::arithmetic::AssignmentFinder_detail::addMVBound (
 const FormulaT & f) [inline]
```

```
0.15.17.2.3 computeCover() Covering smtrat::mcsat::arithmetic::AssignmentFinder_detail::compute←
Cover () [inline]
```

```
0.15.17.2.4 findAssignment() AssignmentOrConflict smtrat::mcsat::arithmetic::AssignmentFinder←
_detail::findAssignment () [inline]
```

## 0.15.18 smtrat::mcsat::smtaf::AssignmentFinder\_SMT Class Reference

```
#include <AssignmentFinder_SMT.h>
```

### Public Member Functions

- `AssignmentFinder_SMT (VariableRange variables, const Model &model)`
- `boost::tribool addConstraint (const FormulaT &f)`
- `boost::tribool addMVBound (const FormulaT &f)`
- `std::optional< AssignmentOrConflict > findAssignment (const VariablePos excludeVar) const`
- `std::optional< AssignmentOrConflict > findAssignment () const`

#### 0.15.18.1 Constructor & Destructor Documentation

**0.15.18.1.1 AssignmentFinder\_SMT()** smtrat::mcsat::smtaf::AssignmentFinder\_SMT::AssignmentFinder\_SMT (

```
 VariableRange variables,
 const Model & model) [inline]
```

#### 0.15.18.2 Member Function Documentation

**0.15.18.2.1 addConstraint()** boost::tribool smtrat::mcsat::smtaf::AssignmentFinder\_SMT::addConstraint (

```
 const FormulaT & f)
```

**0.15.18.2.2 addMVBound()** boost::tribool smtrat::mcsat::smtaf::AssignmentFinder\_SMT::addMVBound (

```
 const FormulaT & f)
```

**0.15.18.2.3 findAssignment() [1/2]** std::optional< AssignmentOrConflict > smtrat::mcsat::smtaf::AssignmentFinder\_SMT::findAssignment ( ) const

**0.15.18.2.4 findAssignment() [2/2]** std::optional< AssignmentOrConflict > smtrat::mcsat::smtaf::AssignmentFinder\_SMT::findAssignment (

```
 const VariablePos excludeVar) const
```

## 0.15.19 smtrat::cadcells::datastructures::detail::AssignmentProperties Struct Reference

```
#include <projections.h>
```

### Data Fields

- `std::map< PolyRef, carl::RealRootsResult< RAN > > real_roots`
- `std::map< PolyRef, bool > is_zero`

#### 0.15.19.1 Field Documentation

**0.15.19.1.1 `is_zero`** std::map<[PolyRef](#), bool> smtrat::cadcells::datastructures::detail::Assignment<Properties::is\_zero

**0.15.19.1.2 `real_roots`** std::map<[PolyRef](#), carl::RealRootsResult<[RAN](#)> > smtrat::cadcells::datastructures<::detail::AssignmentProperties::real\_roots

## 0.15.20 smtrat::parser::Attribute Class Reference

Represents an [Attribute](#).

```
#include <Attribute.h>
```

### Public Types

- using [AttributeValue = types::AttributeValue](#)

### Public Member Functions

- [Attribute \(\)](#)
- [Attribute \(const std::string &key\)](#)
- [Attribute \(const std::string &key, const AttributeValue &value\)](#)
- [Attribute \(const std::string &key, const boost::optional<AttributeValue> &value\)](#)
- [bool hasValue \(\) const](#)
- [void simplify \(\)](#)

### Data Fields

- std::string [key](#)
- [AttributeValue value](#)

#### 0.15.20.1 Detailed Description

Represents an [Attribute](#).

#### 0.15.20.2 Member Typedef Documentation

**0.15.20.2.1 `AttributeValue`** using smtrat::parser::Attribute::AttributeValue = [types::AttributeValue](#)

#### 0.15.20.3 Constructor & Destructor Documentation

**0.15.20.3.1 `Attribute()` [1/4]** smtrat::parser::Attribute::Attribute () [inline]

**0.15.20.3.2 `Attribute()` [2/4]** smtrat::parser::Attribute::Attribute (const std::string & key) [inline], [explicit]

**0.15.20.3.3 `Attribute()` [3/4]** smtrat::parser::Attribute::Attribute (const std::string & key, const AttributeValue & value) [inline]

**0.15.20.3.4 Attribute()** [4/4] smtrat::parser::Attribute::Attribute ( const std::string & key, const boost::optional<AttributeValue> & value ) [inline]

#### 0.15.20.4 Member Function Documentation

**0.15.20.4.1 hasValue()** bool smtrat::parser::Attribute::hasValue ( ) const [inline]

**0.15.20.4.2 simplify()** void smtrat::parser::Attribute::simplify ( ) [inline]

#### 0.15.20.5 Field Documentation

**0.15.20.5.1 key** std::string smtrat::parser::Attribute::key

**0.15.20.5.2 value** AttributeValue smtrat::parser::Attribute::value

### 0.15.21 smtrat::parser::AttributeParser Struct Reference

```
#include <Attribute.h>
```

#### Public Member Functions

- [AttributeParser \(\)](#)

#### Data Fields

- [KeywordParser keyword](#)
- [AttributeValueParser value](#)
- [qi::rule<Iterator, Attribute\(\), Skipper> main](#)

#### 0.15.21.1 Constructor & Destructor Documentation

**0.15.21.1.1 AttributeParser()** smtrat::parser::AttributeParser::AttributeParser ( ) [inline]

#### 0.15.21.2 Field Documentation

**0.15.21.2.1 keyword** KeywordParser smtrat::parser::AttributeParser::keyword

**0.15.21.2.2 main** qi::rule<Iterator, Attribute(), Skipper> smtrat::parser::AttributeParser::main

**0.15.21.2.3 value** AttributeValueParser smtrat::parser::AttributeParser::value

## 0.15.22 smtrat::parser::AttributeValueParser Struct Reference

```
#include <Attribute.h>
```

### Public Types

- `typedef conversion::VariantVariantConverter< types::AttributeValue > Converter`

### Public Member Functions

- `AttributeValueParser ()`

### Data Fields

- `SpecConstantParser speconstant`
- `SymbolParser symbol`
- `SExpressionParser sexpression`
- `Converter converter`
- `qi::rule< Iterator, types::AttributeValue(), Skipper > main`

#### 0.15.22.1 Member Typedef Documentation

##### 0.15.22.1.1 Converter `typedef conversion::VariantVariantConverter<types::AttributeValue> smtrat::parser::AttributeValueParser::Converter`

#### 0.15.22.2 Constructor & Destructor Documentation

##### 0.15.22.2.1 AttributeValueParser() `smtrat::parser::AttributeValueParser::AttributeValueParser ( ) [inline]`

#### 0.15.22.3 Field Documentation

##### 0.15.22.3.1 converter `Converter smtrat::parser::AttributeValueParser::converter`

##### 0.15.22.3.2 main `qi::rule<Iterator, types::AttributeValue(), Skipper> smtrat::parser::AttributeValueParser::main`

##### 0.15.22.3.3 sexpression `SExpressionParser smtrat::parser::AttributeValueParser::sexpression`

##### 0.15.22.3.4 speconstant `SpecConstantParser smtrat::parser::AttributeValueParser::speconstant`

##### 0.15.22.3.5 symbol `SymbolParser smtrat::parser::AttributeValueParser::symbol`

## 0.15.23 smtrat::AxiomFactory Class Reference

```
#include <AxiomFactory.h>
```

## Public Types

- enum `AxiomType` {
 `ZERO` , `TANGENT_PLANE` , `MONOTONICITY` , `CONGRUENCE` ,
 `ICP` }

## Static Public Member Functions

- static `FormulasT createFormula (AxiomType axiomType, Model linearizedModel, MonomialMap monomialMap)`

### 0.15.23.1 Member Enumeration Documentation

#### 0.15.23.1.1 AxiomType enum `smtrat::AxiomFactory::AxiomType`

Enumerator

|                            |  |
|----------------------------|--|
| <code>ZERO</code>          |  |
| <code>TANGENT_PLANE</code> |  |
| <code>MONOTONICITY</code>  |  |
| <code>CONGRUENCE</code>    |  |
| <code>ICP</code>           |  |

### 0.15.23.2 Member Function Documentation

#### 0.15.23.2.1 `createFormula()` `FormulasT smtrat::AxiomFactory::createFormula (AxiomType axiomType, Model linearizedModel, MonomialMap monomialMap) [static]`

## 0.15.24 benchmax::Backend Class Reference

Base class for all backends.

```
#include <Backend.h>
```

## Public Member Functions

- bool `suspendable () const`
- void `process_results (const Jobs &jobs, bool check_finished)`
- void `addResult (const Tool *tool, const fs::path &file, BenchmarkResult &&result)`

*Add a result.*
- void `run (const Jobs &jobs, bool wait_for_termination)`

*Run the list of tools against the list of benchmarks.*

## Protected Member Functions

- `Backend ()`
- virtual void `startTool (const Tool *)`

*Hook for every tool at the beginning.*
- virtual void `finalize ()`

*Hook to allow for asynchronous backends to wait for jobs to terminate.*
- virtual void `execute (const Tool *, const fs::path &)`

- Execute a single pair of tool and benchmark.*
- void [madeProgress](#) (std::size\_t files=1)  
*Can be called to give information about the current progress, if available.*
  - virtual bool [collect\\_results](#) (const [Jobs](#) &, bool)
  - void [sanitize\\_results](#) (const [Jobs](#) &jobs) const
  - void [write\\_results](#) (const [Jobs](#) &jobs) const

## Protected Attributes

- std::size\_t [mExpectedJobs](#)  
*Number of jobs that should be run.*
- std::atomic< std::size\_t > [mFinishedJobs](#)  
*Number of jobs that are finished.*
- std::atomic< std::size\_t > [mLastPercent](#)  
*Percentage of finished jobs when [madeProgress\(\)](#) was last called.*

### 0.15.24.1 Detailed Description

Base class for all backends.

Offers appropriate hooks to model the whole workflow for backends where each job is executed individually. The [run\(\)](#) method is called from outside which first calls [startTool\(\)](#) for every tool and then runs [execute\(\)](#) for every pair of tool and benchmark. It also offers [addResult\(\)](#) to store results and [madeProgress\(\)](#) to provide a progress indication to the user. If a benchmark requires a completely different workflow, for example for a batch job, it should override the [run\(\)](#) method.

### 0.15.24.2 Constructor & Destructor Documentation

#### 0.15.24.2.1 [Backend\(\)](#) `benchmax::Backend::Backend( ) [inline], [protected]`

### 0.15.24.3 Member Function Documentation

#### 0.15.24.3.1 [addResult\(\)](#) `void benchmax::Backend::addResult( const Tool * tool, const fs::path & file, BenchmarkResult && result ) [inline]`

Add a result.

#### 0.15.24.3.2 [collect\\_results\(\)](#) `virtual bool benchmax::Backend::collect_results( const Jobs &, bool ) [inline], [protected], [virtual]`

#### 0.15.24.3.3 [execute\(\)](#) `virtual void benchmax::Backend::execute( const Tool *, const fs::path & ) [inline], [protected], [virtual]`

Execute a single pair of tool and benchmark.

Reimplemented in [benchmax::LocalBackend](#), and [benchmax::CondorBackend](#).

#### 0.15.24.3.4 [finalize\(\)](#) `virtual void benchmax::Backend::finalize( ) [inline], [protected], [virtual]`

Hook to allow for asynchronous backends to wait for jobs to terminate.

**0.15.24.3.5 madeProgress()** void benchmax::Backend::madeProgress (

std::size\_t files = 1 ) [inline], [protected]

Can be called to give information about the current progress, if available.

**0.15.24.3.6 process\_results()** void benchmax::Backend::process\_results (

const Jobs &amp; jobs,

bool check\_finished ) [inline]

**0.15.24.3.7 run()** void benchmax::Backend::run (

const Jobs &amp; jobs,

bool wait\_for\_termination ) [inline]

Run the list of tools against the list of benchmarks.

**0.15.24.3.8 sanitize\_results()** void benchmax::Backend::sanitize\_results (

const Jobs &amp; jobs ) const [inline], [protected]

**0.15.24.3.9 startTool()** virtual void benchmax::Backend::startTool (

const Tool \* ) [inline], [protected], [virtual]

Hook for every tool at the beginning.

Can be used to upload the tool to some remote system.

**0.15.24.3.10 suspendable()** bool benchmax::Backend::suspendable ( ) const [inline]**0.15.24.3.11 write\_results()** void benchmax::Backend::write\_results (

const Jobs &amp; jobs ) const [inline], [protected]

**0.15.24.4 Field Documentation****0.15.24.4.1 mExpectedJobs** std::size\_t benchmax::Backend::mExpectedJobs [protected]

Number of jobs that should be run.

**0.15.24.4.2 mFinishedJobs** std::atomic<std::size\_t> benchmax::Backend::mFinishedJobs [protected]

Number of jobs that are finished.

**0.15.24.4.3 mLastPercent** std::atomic<std::size\_t> benchmax::Backend::mLastPercent [protected]Percentage of finished jobs when [madeProgress\(\)](#) was last called.**0.15.25 smrat::Backend< Settings > Class Template Reference**

#include &lt;Backend.h&gt;

**Public Member Functions**

- [Backend \(\)](#)
- void [init \(std::vector< carl::Variable > &varOrdering\)](#)
- std::size\_t [dimension \(\)](#)
- std::shared\_ptr< cadcells::Polynomial::ContextType > [getContext \(\)](#)

- const carl::Assignment< cadcells::RAN > & `getCurrentAssignment ()`
- auto & `getCoveringInformation ()`
- `FormulaSetT getInfeasibleSubset ()`
- void `addConstraint (const ConstraintT &constraint)`
- void `addConstraint (const size_t level, const std::vector< ConstraintT > &&constraints)`
- auto & `getUnknownConstraints ()`
- auto & `getUnknownConstraints (std::size_t &level)`
- auto & `getKnownConstraints ()`
- auto & `getKnownConstraints (std::size_t &level)`
- void `updateAssignment (std::size_t level)`
- void `resetStoredData (std::size_t level)`
- void `resetDerivationToConstraintMap ()`
- void `removeConstraint (const ConstraintT &constraint)`
- void `setConstraintsKnown (const std::size_t &level)`
- void `setConstraintsUnknown (const std::size_t &level)`
- void `processUnknownConstraints (const std::size_t &level, const bool prune_assignment)`
- `Answer getUnsatCover (const std::size_t level)`

### 0.15.25.1 Constructor & Destructor Documentation

**0.15.25.1.1 Backend()** template<typename Settings >  
void `smrat::Backend< Settings >::Backend ()` [inline]

### 0.15.25.2 Member Function Documentation

**0.15.25.2.1 addConstraint() [1/2]** template<typename Settings >  
void `smrat::Backend< Settings >::addConstraint (`  
    const `ConstraintT & constraint )` [inline]

**0.15.25.2.2 addConstraint() [2/2]** template<typename Settings >  
void `smrat::Backend< Settings >::addConstraint (`  
    const `size_t level,`  
    const `std::vector< ConstraintT > && constraints )` [inline]

**0.15.25.2.3 dimension()** template<typename Settings >  
`std::size_t smrat::Backend< Settings >::dimension ()` [inline]

**0.15.25.2.4 getContext()** template<typename Settings >  
`std::shared_ptr<cadcells::Polynomial::ContextType> smrat::Backend< Settings >::getContext (`  
    `)` [inline]

**0.15.25.2.5 getCoveringInformation()** template<typename Settings >  
auto& `smrat::Backend< Settings >::getCoveringInformation ()` [inline]

**0.15.25.2.6 getCurrentAssignment()** template<typename Settings >  
const carl::Assignment<cadcells::RAN>& `smrat::Backend< Settings >::getCurrentAssignment ()`  
[inline]

**0.15.25.2.7 `getInfeasibleSubset()`** template<typename Settings >  
FormulaSetT smtrat::Backend< Settings >::getInfeasibleSubset ( ) [inline]

**0.15.25.2.8 `getKnownConstraints()` [1/2]** template<typename Settings >  
auto& smtrat::Backend< Settings >::getKnownConstraints ( ) [inline]

**0.15.25.2.9 `getKnownConstraints()` [2/2]** template<typename Settings >  
auto& smtrat::Backend< Settings >::getKnownConstraints ( std::size\_t & level ) [inline]

**0.15.25.2.10 `getUnknownConstraints()` [1/2]** template<typename Settings >  
auto& smtrat::Backend< Settings >::getUnknownConstraints ( ) [inline]

**0.15.25.2.11 `getUnknownConstraints()` [2/2]** template<typename Settings >  
auto& smtrat::Backend< Settings >::getUnknownConstraints ( std::size\_t & level ) [inline]

**0.15.25.2.12 `getUnsatCover()`** template<typename Settings >  
Answer smtrat::Backend< Settings >::getUnsatCover ( const std::size\_t level ) [inline]

**0.15.25.2.13 `init()`** template<typename Settings >  
void smtrat::Backend< Settings >::init ( std::vector< carl::Variable > & varOrdering ) [inline]

**0.15.25.2.14 `processUnknownConstraints()`** template<typename Settings >  
void smtrat::Backend< Settings >::processUnknownConstraints ( const std::size\_t & level, const bool prune\_assignment ) [inline]

**0.15.25.2.15 `removeConstraint()`** template<typename Settings >  
void smtrat::Backend< Settings >::removeConstraint ( const ConstraintT & constraint ) [inline]

**0.15.25.2.16 `resetDerivationToConstraintMap()`** template<typename Settings >  
void smtrat::Backend< Settings >::resetDerivationToConstraintMap ( ) [inline]

**0.15.25.2.17 `resetStoredData()`** template<typename Settings >  
void smtrat::Backend< Settings >::resetStoredData ( std::size\_t level ) [inline]

**0.15.25.2.18 `setConstraintsKnown()`** template<typename Settings >  
void smtrat::Backend< Settings >::setConstraintsKnown ( const std::size\_t & level ) [inline]

```
0.15.25.2.19 setConstraintsUnknown() template<typename Settings >
void smtrat::Backend< Settings >::setConstraintsUnknown (
 const std::size_t & level) [inline]
```

```
0.15.25.2.20 updateAssignment() template<typename Settings >
void smtrat::Backend< Settings >::updateAssignment (
 std::size_t level) [inline]
```

## 0.15.26 smtrat::BackendLink Class Reference

```
#include <StrategyGraph.h>
```

### Public Member Functions

- `BackendLink (std::size_t target, std::size_t priority, const ConditionFunction &cf)`
- `bool checkCondition (const carl::Condition &c) const`
- `std::size_t getTarget () const`
- `std::size_t getPriority () const`
- `bool operator< (const BackendLink &rhs) const`
- `BackendLink & priority (std::size_t p)`
- `template<typename T >
 BackendLink & condition (const T &f)`
- `BackendLink & id (std::size_t &id)`

### 0.15.26.1 Constructor & Destructor Documentation

```
0.15.26.1.1 BackendLink() smtrat::BackendLink::BackendLink (
 std::size_t target,
 std::size_t priority,
 const ConditionFunction & cf) [inline]
```

### 0.15.26.2 Member Function Documentation

```
0.15.26.2.1 checkCondition() bool smtrat::BackendLink::checkCondition (
 const carl::Condition & c) const [inline]
```

```
0.15.26.2.2 condition() template<typename T >
BackendLink& smtrat::BackendLink::condition (
 const T & f) [inline]
```

```
0.15.26.2.3 getPriority() std::size_t smtrat::BackendLink::getPriority () const [inline]
```

```
0.15.26.2.4 getTarget() std::size_t smtrat::BackendLink::getTarget () const [inline]
```

```
0.15.26.2.5 id() BackendLink& smtrat::BackendLink::id (
 std::size_t & id) [inline]
```

**0.15.26.2.6 operator<()** `bool smtrat::BackendLink::operator< ( const BackendLink & rhs ) const [inline]`

**0.15.26.2.7 priority()** `BackendLink& smtrat::BackendLink::priority ( std::size_t p ) [inline]`

## 0.15.27 smtrat::BackendSynchronisation Class Reference

#include <ThreadPool.h>

### Public Member Functions

- `BackendSynchronisation ()`
- `void wait ()`
- `void notify ()`

#### 0.15.27.1 Constructor & Destructor Documentation

**0.15.27.1.1 BackendSynchronisation()** `smtrat::BackendSynchronisation::BackendSynchronisation ( ) [inline]`

#### 0.15.27.2 Member Function Documentation

**0.15.27.2.1 notify()** `void smtrat::BackendSynchronisation::notify ( ) [inline]`

**0.15.27.2.2 wait()** `void smtrat::BackendSynchronisation::wait ( ) [inline]`

## 0.15.28 smtrat::Backtrackable< UnionFind > Struct Template Reference

#include <UnionFind.h>

### Public Types

- using `Value` = typename UnionFind::Value
- using `Representative` = `Value`
- using `Timestamp` = std::pair< `Value`, `Value` >
- using `History` = std::vector< std::pair< `Timestamp`, UnionFind > >

### Public Member Functions

- `Backtrackable ()`
- `void init_size (size_t size) noexcept`
- `auto find (Value const &val) noexcept -> Representative`
- `void merge (Value const &a, Value const &b) noexcept`
- `void backtrack (Value const &a, Value const &b) noexcept`
- `auto version (Timestamp &&ts) const noexcept -> typename History::const_iterator`
- `constexpr auto origin () const noexcept -> Timestamp`
- `UnionFind & current () noexcept`

### Data Fields

- `History history`

### 0.15.28.1 Member Typedef Documentation

**0.15.28.1.1 History** template<typename UnionFind >  
using smtrat::Backtrackable< UnionFind >::History = std::vector< std::pair< Timestamp, UnionFind > >

**0.15.28.1.2 Representative** template<typename UnionFind >  
using smtrat::Backtrackable< UnionFind >::Representative = Value

**0.15.28.1.3 Timestamp** template<typename UnionFind >  
using smtrat::Backtrackable< UnionFind >::Timestamp = std::pair<Value, Value>

**0.15.28.1.4 Value** template<typename UnionFind >  
using smtrat::Backtrackable< UnionFind >::Value = typename UnionFind::Value

### 0.15.28.2 Constructor & Destructor Documentation

**0.15.28.2.1 Backtrackable()** template<typename UnionFind >  
smtrat::Backtrackable< UnionFind >::Backtrackable ( ) [inline]

### 0.15.28.3 Member Function Documentation

**0.15.28.3.1 backtrack()** template<typename UnionFind >  
void smtrat::Backtrackable< UnionFind >::backtrack (  
 Value const & a,  
 Value const & b ) [inline], [noexcept]

**0.15.28.3.2 current()** template<typename UnionFind >  
UnionFind& smtrat::Backtrackable< UnionFind >::current ( ) [inline], [noexcept]

**0.15.28.3.3 find()** template<typename UnionFind >  
auto smtrat::Backtrackable< UnionFind >::find (  
 Value const & val ) -> Representative [inline], [noexcept]

**0.15.28.3.4 init\_size()** template<typename UnionFind >  
void smtrat::Backtrackable< UnionFind >::init\_size (  
 size\_t size ) [inline], [noexcept]

**0.15.28.3.5 merge()** template<typename UnionFind >  
void smtrat::Backtrackable< UnionFind >::merge (  
 Value const & a,  
 Value const & b ) [inline], [noexcept]

```
0.15.28.3.6 origin() template<typename UnionFind >
constexpr auto smtrat::Backtrackable< UnionFind >::origin () const -> Timestamp [inline],
[constexpr], [noexcept]
```

```
0.15.28.3.7 version() template<typename UnionFind >
auto smtrat::Backtrackable< UnionFind >::version (
 Timestamp && ts) const -> typename History::const_iterator [inline], [noexcept]
```

#### 0.15.28.4 Field Documentation

```
0.15.28.4.1 history template<typename UnionFind >
History smtrat::Backtrackable< UnionFind >::history
```

### 0.15.29 smtrat::cadcells::datastructures::BaseDerivation< Properties > Class Template Reference

A [BaseDerivation](#) has a level and a set of properties of this level, and an underlying derivation representing the lower levels.

```
#include <derivation.h>
```

#### Public Member Functions

- [BaseDerivation \(Projections &projections, DerivationRef< Properties > underlying, size\\_t level\)](#)
- [BaseDerivation \(const BaseDerivation &other\)](#)
- [DerivationRef< Properties > & underlying \(\)](#)
- [const DerivationRef< Properties > & underlying \(\) const](#)
- [PolyPool & polys \(\)](#)
- [Projections & proj \(\)](#)
- [carl::Variable main\\_var \(\) const](#)
- [size\\_t level \(\) const](#)
- [template<typename P > void insert \(P property\)](#)
- [template<typename P > bool contains \(const P &property\) const](#)
- [template<typename P > const PropertiesTSet< P > & properties \(\) const](#)
- [void merge\\_with \(const BaseDerivation< Properties > &other\)](#)

#### Friends

- [template<typename P > void merge\\_underlying \(std::vector< SampledDerivationRef< P > > &derivations\)](#)

#### 0.15.29.1 Detailed Description

```
template<typename Properties>
class smtrat::cadcells::datastructures::BaseDerivation< Properties >
```

A [BaseDerivation](#) has a level and a set of properties of this level, and an underlying derivation representing the lower levels.

#### Template Parameters

|                   |                                       |
|-------------------|---------------------------------------|
| <i>Properties</i> | Set of properties (from the operator) |
|-------------------|---------------------------------------|

## 0.15.29.2 Constructor & Destructor Documentation

**0.15.29.2.1 BaseDerivation() [1/2]** template<typename Properties >  
smtrat::cadcells::datastructures::BaseDerivation< Properties >::BaseDerivation (  
    Projections & *projections*,  
    DerivationRef< Properties > *underlying*,  
    size\_t *level*) [inline]

**0.15.29.2.2 BaseDerivation() [2/2]** template<typename Properties >  
smtrat::cadcells::datastructures::BaseDerivation< Properties >::BaseDerivation (  
    const BaseDerivation< Properties > & *other*) [inline]

## 0.15.29.3 Member Function Documentation

**0.15.29.3.1 contains()** template<typename Properties >  
template<typename P >  
bool smtrat::cadcells::datastructures::BaseDerivation< Properties >::contains (  
    const P & *property*) const [inline]

**0.15.29.3.2 insert()** template<typename Properties >  
template<typename P >  
void smtrat::cadcells::datastructures::BaseDerivation< Properties >::insert (  
    P *property*) [inline]

**0.15.29.3.3 level()** template<typename Properties >  
size\_t smtrat::cadcells::datastructures::BaseDerivation< Properties >::level () const [inline]

**0.15.29.3.4 main\_var()** template<typename Properties >  
carl::Variable smtrat::cadcells::datastructures::BaseDerivation< Properties >::main\_var ()  
const [inline]

**0.15.29.3.5 merge\_with()** template<typename Properties >  
void smtrat::cadcells::datastructures::BaseDerivation< Properties >::merge\_with (  
    const BaseDerivation< Properties > & *other*) [inline]

**0.15.29.3.6 polys()** template<typename Properties >  
PolyPool& smtrat::cadcells::datastructures::BaseDerivation< Properties >::polys () [inline]

**0.15.29.3.7 proj()** template<typename Properties >  
Projections& smtrat::cadcells::datastructures::BaseDerivation< Properties >::proj () [inline]

```
0.15.29.3.8 properties() template<typename Properties >
template<typename P >
const PropertiesTSet<P>& smtrat::cadcells::datastructures::BaseDerivation< Properties >↔
::properties () const [inline]
```

```
0.15.29.3.9 underlying() [1/2] template<typename Properties >
DerivationRef<Properties>& smtrat::cadcells::datastructures::BaseDerivation< Properties >↔
::underlying () [inline]
```

```
0.15.29.3.10 underlying() [2/2] template<typename Properties >
const DerivationRef<Properties>& smtrat::cadcells::datastructures::BaseDerivation< Properties >::underlying () const [inline]
```

#### 0.15.29.4 Friends And Related Function Documentation

```
0.15.29.4.1 merge_underlying template<typename Properties >
template<typename P >
void merge_underlying (
 std::vector< SampledDerivationRef< P >> & derivations) [friend]
```

### 0.15.30 smtrat::cad::BaseProjection< Settings > Class Template Reference

```
#include <BaseProjection.h>
```

#### Public Member Functions

- std::size\_t **dim** () const  
*Returns the dimension of the projection.*
- const auto & **vars** () const  
*Returns the variables used for projection.*
- void **reset** ()  
*Resets all datastructures, use the given variables from now on.*
- template<typename F >  
 void **setRemoveCallback** (F &&f)  
*Sets a callback that is called whenever polynomials are removed.*
- carl::Bitset **addPolynomial** (const **Poly** &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual carl::Bitset **addPolynomial** (const **UPoly** &p, std::size\_t cid, bool isBound)=0  
*Adds the given polynomial to the projection.*
- carl::Bitset **addEqConstraint** (const **Poly** &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual carl::Bitset **addEqConstraint** (const **UPoly** &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial of an equational constraint to the projection.*
- void **removePolynomial** (const **Poly** &p, std::size\_t cid, bool isBound)  
*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual void **removePolynomial** (const **UPoly** &p, std::size\_t cid, bool isBound)=0  
*Removes the given polynomial from the projection.*
- virtual std::size\_t **size** (std::size\_t level) const =0
- std::size\_t **size** () const
- virtual bool **empty** (std::size\_t level) const =0

- virtual bool `empty ()`
- `OptionalID getPolyForLifting (std::size_t level, SampleLiftedWith &slw)`  
*Get a polynomial from this level suited for lifting.*
- virtual bool `hasPolynomialById (std::size_t level, std::size_t id) const =0`
- virtual const `UPoly & getPolynomialById (std::size_t level, std::size_t id) const =0`  
*Retrieves a polynomial from its id.*
- virtual void `exportAsDot (std::ostream &) const`
- virtual `Origin getOrigin (std::size_t level, std::size_t id) const`

### Protected Types

- using `Constraints = CADConstraints< Settings::backtracking >`

### Protected Member Functions

- `BaseProjection (const Constraints &c)`
- void `callRemoveCallback (std::size_t level, const SampleLiftedWith &slw) const`
- `std::size_t getID (std::size_t level)`  
*Returns a fresh polynomial id for the given level.*
- void `freeID (std::size_t level, std::size_t id)`  
*Frees a currently used polynomial id for the given level.*
- `carl::Variable var (std::size_t level) const`  
*Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
- bool `canBePurgedByBounds (const UPoly &p) const`  
*Checks whether a polynomial can safely be ignored due to the bounds.*
- bool `isPurged (std::size_t level, std::size_t id)`

### Protected Attributes

- const `Constraints & mConstraints`
- `std::vector< PolynomialLiftingQueue< BaseProjection > > mLiftingQueues`  
*List of lifting queues that can be used for incremental projection.*
- `ProjectionInformation mInfo`  
*Additional info on projection, projection levels and projection polynomials.*
- `ProjectionOperator mOperator`  
*The projection operator.*
- `std::function< void(std::size_t, const SampleLiftedWith &) > mRemoveCallback`  
*Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

#### 0.15.30.1 Member Typedef Documentation

```
0.15.30.1.1 Constraints template<typename Settings >
using smrat::cad::BaseProjection< Settings >::Constraints = CADConstraints<Settings::backtracking>
[protected]
```

#### 0.15.30.2 Constructor & Destructor Documentation

```
0.15.30.2.1 BaseProjection() template<typename Settings >
smrat::cad::BaseProjection< Settings >::BaseProjection (
 const Constraints & c) [inline], [protected]
```

### 0.15.30.3 Member Function Documentation

**0.15.30.3.1 addEqConstraint() [1/2]** template<typename Settings >  
 carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint ( const Poly & p, std::size\_t cid, bool isBound ) [inline]

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.30.3.2 addEqConstraint() [2/2]** template<typename Settings >  
 virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint ( const UPoly & p, std::size\_t cid, bool isBound ) [inline], [virtual]

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >](#).

**0.15.30.3.3 addPolynomial() [1/2]** template<typename Settings >  
 carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial ( const Poly & p, std::size\_t cid, bool isBound ) [inline]

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.30.3.4 addPolynomial() [2/2]** template<typename Settings >  
 virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial ( const UPoly & p, std::size\_t cid, bool isBound ) [pure virtual]

Adds the given polynomial to the projection.

Implemented in [smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >](#), [smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >](#), [smtrat::cad::Projection< Incrementality::INCREMENTAL, Backtracking::HIDE, Settings >](#), [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >](#), and [smtrat::cad::Projection< Incrementality::SIMPLE, Backtracking::HIDE, Settings >](#).

**0.15.30.3.5 callRemoveCallback()** template<typename Settings >  
 void smtrat::cad::BaseProjection< Settings >::callRemoveCallback ( std::size\_t level, const SampleLiftedWith & slw ) const [inline], [protected]

**0.15.30.3.6 canBePurgedByBounds()** template<typename Settings >  
 bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds ( const UPoly & p ) const [inline], [protected]

Checks whether a polynomial can safely be ignored due to the bounds.

**0.15.30.3.7 dim()** template<typename Settings >  
 std::size\_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline]

Returns the dimension of the projection.

**0.15.30.3.8 empty() [1/2]** template<typename Settings >  
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual]

**0.15.30.3.9 empty() [2/2]** template<typename Settings >  
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( std::size\_t level ) const [pure virtual]

Implemented in [smtrat::qe::cad::Projection< Settings >](#), [smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >](#), [smtrat::cad::ModelBasedProjection< Incrementality::NONE, smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#), and [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDDEN >](#).

**0.15.30.3.10 exportAsDot()** template<typename Settings >  
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot ( std::ostream & ) const [inline], [virtual]

Reimplemented in [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#), and [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDDEN >](#).

**0.15.30.3.11 freeID()** template<typename Settings >  
void smtrat::cad::BaseProjection< Settings >::freeID ( std::size\_t level, std::size\_t id ) [inline], [protected]

Frees a currently used polynomial id for the given level.

**0.15.30.3.12 getID()** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::getID ( std::size\_t level ) [inline], [protected]

Returns a fresh polynomial id for the given level.

**0.15.30.3.13 getOrigin()** template<typename Settings >  
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin ( std::size\_t level, std::size\_t id ) const [inline], [virtual]

Reimplemented in [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#).

**0.15.30.3.14 getPolyForLifting()** template<typename Settings >  
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting ( std::size\_t level, SampleLiftedWith & slw ) [inline]

Get a polynomial from this level suited for lifting.

**0.15.30.3.15 getPolynomialById()** template<typename Settings >  
virtual const UPoly& smtrat::cad::BaseProjection< Settings >::getPolynomialById ( std::size\_t level, std::size\_t id ) const [pure virtual]

Retrieves a polynomial from its id.

Implemented in [smtrat::qe::cad::Projection< Settings >](#), [smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >](#), [smtrat::cad::ModelBasedProjection< Incrementality::NONE, smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#), and [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDDEN >](#).

```
0.15.30.3.16 hasPolynomialById() template<typename Settings >
virtual bool smtrat::cad::BaseProjection< Settings >::hasPolynomialById (
 std::size_t level,
 std::size_t id) const [pure virtual]
```

Implemented in `smtrat::qe::cad::Projection< Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`

```
0.15.30.3.17 isPurged() template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::isPurged (
 std::size_t level,
 std::size_t id) [inline], [protected]
```

```
0.15.30.3.18 removePolynomial() [1/2] template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.30.3.19 removePolynomial() [2/2] template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::removePolynomial (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [pure virtual]
```

Removes the given polynomial from the projection.

Implemented in `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::NONE, Backtracking::HIDE, Settings >`

```
0.15.30.3.20 reset() template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::reset () [inline]
```

Resets all datastructures, use the given variables from now on.

```
0.15.30.3.21 setRemoveCallback() template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
 F && f) [inline]
```

Sets a callback that is called whenever polynomials are removed.

```
0.15.30.3.22 size() [1/2] template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size () const [inline]
```

```
0.15.30.3.23 size() [2/2] template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
 std::size_t level) const [pure virtual]
```

Implemented in `smtrat::qe::cad::Projection< Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`

```
0.15.30.3.24 var() template<typename Settings >
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
 std::size_t level) const [inline], [protected]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

```
0.15.30.3.25 vars() template<typename Settings >
const auto& smtrat::cad::BaseProjection< Settings >::vars () const [inline]
```

Returns the variables used for projection.

#### 0.15.30.4 Field Documentation

```
0.15.30.4.1 mConstraints template<typename Settings >
const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected]
```

```
0.15.30.4.2 mInfo template<typename Settings >
ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected]
```

Additional info on projection, projection levels and projection polynomials.

```
0.15.30.4.3 mLiftingQueues template<typename Settings >
std::vector<PolynomialLiftingQueue<BaseProjection> > smtrat::cad::BaseProjection< Settings
>::mLiftingQueues [protected]
```

List of lifting queues that can be used for incremental projection.

```
0.15.30.4.4 mOperator template<typename Settings >
ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator [protected]
```

The projection operator.

```
0.15.30.4.5 mRemoveCallback template<typename Settings >
std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings
>::mRemoveCallback [protected]
```

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

### 0.15.31 smtrat::cad::BaseSettings Struct Reference

```
#include <Settings.h>
```

#### Static Public Attributes

- static constexpr Incrementality incrementality = Incrementality::NONE
- static constexpr Backtracking backtracking = Backtracking::ORDERED
- static constexpr ProjectionType projectionOperator = cad::ProjectionType::McCallum
- static constexpr CoreHeuristic coreHeuristic = cad::CoreHeuristic::PreferProjection
- static constexpr MISHeuristic misHeuristic = cad::MISHeuristic::GREEDY
- static constexpr std::size\_t trivialSampleRadius = 1
- static constexpr bool simplifyProjectionByBounds = true
- static constexpr ProjectionCompareStrategy projectionComparator = cad::ProjectionCompareStrategy::Default
- static constexpr SampleCompareStrategy sampleComparator = cad::SampleCompareStrategy::Default
- static constexpr FullSampleCompareStrategy fullSampleComparator = cad::FullSampleCompareStrategy::Default

### 0.15.31.1 Field Documentation

**0.15.31.1.1 backtracking** constexpr `Backtracking` smtrat::cad::BaseSettings::backtracking = `Backtracking::ORDERED` [static], [constexpr]

**0.15.31.1.2 coreHeuristic** constexpr `CoreHeuristic` smtrat::cad::BaseSettings::coreHeuristic = `cad::CoreHeuristic::PreferProjection` [static], [constexpr]

**0.15.31.1.3 fullSampleComparator** constexpr `FullSampleCompareStrategy` smtrat::cad::BaseSettings::fullSampleComparator = `cad::FullSampleCompareStrategy::Default` [static], [constexpr]

**0.15.31.1.4 incrementality** constexpr `Incrementality` smtrat::cad::BaseSettings::incrementality = `Incrementality::NONE` [static], [constexpr]

**0.15.31.1.5 misHeuristic** constexpr `MISHeuristic` smtrat::cad::BaseSettings::misHeuristic = `cad::MISHeuristic::GEOMETRIC` [static], [constexpr]

**0.15.31.1.6 projectionComparator** constexpr `ProjectionCompareStrategy` smtrat::cad::BaseSettings::projectionComparator = `cad::ProjectionCompareStrategy::Default` [static], [constexpr]

**0.15.31.1.7 projectionOperator** constexpr `ProjectionType` smtrat::cad::BaseSettings::projectionOperator = `cad::ProjectionType::McCallum` [static], [constexpr]

**0.15.31.1.8 sampleComparator** constexpr `SampleCompareStrategy` smtrat::cad::BaseSettings::sampleComparator = `cad::SampleCompareStrategy::Default` [static], [constexpr]

**0.15.31.1.9 simplifyProjectionByBounds** constexpr bool smtrat::cad::BaseSettings::simplifyProjectionByBounds = true [static], [constexpr]

**0.15.31.1.10 trivialSampleRadius** constexpr std::size\_t smtrat::cad::BaseSettings::trivialSampleRadius = 1 [static], [constexpr]

## 0.15.32 smtrat::expression::simplifier::BaseSimplifier Struct Reference

```
#include <BaseSimplifier.h>
```

### Public Member Functions

- const `ExpressionContent * operator()` (const `carl::Variable &expr`) const
- const `ExpressionContent * operator()` (const `ITEExpression &expr`) const
- const `ExpressionContent * operator()` (const `QuantifierExpression &expr`) const
- const `ExpressionContent * operator()` (const `UnaryExpression &expr`) const
- const `ExpressionContent * operator()` (const `BinaryExpression &expr`) const
- const `ExpressionContent * operator()` (const `NaryExpression &expr`) const
- const `ExpressionContent * operator()` (const `ExpressionContent *_ec`) const

## Protected Member Functions

- virtual const `ExpressionContent * simplify` (const `carl::Variable &`) const
- virtual const `ExpressionContent * simplify` (const `ITEExpression &`) const
- virtual const `ExpressionContent * simplify` (const `QuantifierExpression &`) const
- virtual const `ExpressionContent * simplify` (const `UnaryExpression &`) const
- virtual const `ExpressionContent * simplify` (const `BinaryExpression &`) const
- virtual const `ExpressionContent * simplify` (const `NaryExpression &`) const

### 0.15.32.1 Member Function Documentation

**0.15.32.1.1 operator() [1/7]** const `ExpressionContent* smtrat::expression::simplifier::Base<Simplifier>::operator()` (const `BinaryExpression & expr`) const [inline]

**0.15.32.1.2 operator() [2/7]** const `ExpressionContent* smtrat::expression::simplifier::Base<Simplifier>::operator()` (const `carl::Variable & expr`) const [inline]

**0.15.32.1.3 operator() [3/7]** const `ExpressionContent* smtrat::expression::simplifier::Base<Simplifier>::operator()` (const `ExpressionContent * _ec`) const [inline]

**0.15.32.1.4 operator() [4/7]** const `ExpressionContent* smtrat::expression::simplifier::Base<Simplifier>::operator()` (const `ITEExpression & expr`) const [inline]

**0.15.32.1.5 operator() [5/7]** const `ExpressionContent* smtrat::expression::simplifier::Base<Simplifier>::operator()` (const `NaryExpression & expr`) const [inline]

**0.15.32.1.6 operator() [6/7]** const `ExpressionContent* smtrat::expression::simplifier::Base<Simplifier>::operator()` (const `QuantifierExpression & expr`) const [inline]

**0.15.32.1.7 operator() [7/7]** const `ExpressionContent* smtrat::expression::simplifier::Base<Simplifier>::operator()` (const `UnaryExpression & expr`) const [inline]

**0.15.32.1.8 simplify() [1/6]** virtual const `ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify` (const `BinaryExpression &`) const [inline], [protected], [virtual]

**0.15.32.1.9 simplify() [2/6]** virtual const `ExpressionContent* smtrat::expression::simplifier::BaseSimplifier::simplify` (const `carl::Variable &`) const [inline], [protected], [virtual]

**0.15.32.1.10 `simplify()` [3/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier:: $\leftarrow$  BaseSimplifier::simplify ( const `ITEExpression` & ) const [inline], [protected], [virtual]

**0.15.32.1.11 `simplify()` [4/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier:: $\leftarrow$  BaseSimplifier::simplify ( const `NaryExpression` & ) const [inline], [protected], [virtual]

Reimplemented in `smtrat::expression::simplifier::SingletonSimplifier`, `smtrat::expression::simplifier::MergeSimplifier`, and `smtrat::expression::simplifier::DuplicateSimplifier`.

**0.15.32.1.12 `simplify()` [5/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier:: $\leftarrow$  BaseSimplifier::simplify ( const `QuantifierExpression` & ) const [inline], [protected], [virtual]

**0.15.32.1.13 `simplify()` [6/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier:: $\leftarrow$  BaseSimplifier::simplify ( const `UnaryExpression` & ) const [inline], [protected], [virtual]

Reimplemented in `smtrat::expression::simplifier::NegationSimplifier`.

## 0.15.33 smtrat::cad::Origin::BaseType Struct Reference

```
#include <Origin.h>
```

### Public Member Functions

- `BaseType` (std::size\_t `level`, std::size\_t `id`)
- `BaseType` (std::size\_t `lvl`, std::size\_t `id1`, std::size\_t `id2`)
- bool `active` () const
- void `activate` (const carl::Bitset &`ids`)
- void `deactivate` (const carl::Bitset &`ids`)
- bool `operator==` (const `BaseType` &`bt`) const
- bool `operator<` (const `BaseType` &`bt`) const

### Data Fields

- std::size\_t `level`
- std::size\_t `first`
- std::size\_t `second`
- bool `first_active` = true
- bool `second_active` = true
- bool `ec_active` = true

### Friends

- std::ostream & `operator<<` (std::ostream &`os`, const `BaseType` &`bt`)

### 0.15.33.1 Constructor & Destructor Documentation

**0.15.33.1.1 `BaseType()` [1/2]** smtrat::cad::Origin::BaseType::`BaseType` ( std::size\_t `level`, std::size\_t `id` ) [inline], [explicit]

**0.15.33.1.2 `BaseType()` [2/2]** smtrat::cad::Origin::BaseType::BaseType ( std::size\_t *lvl*, std::size\_t *id1*, std::size\_t *id2* ) [inline]

### 0.15.33.2 Member Function Documentation

**0.15.33.2.1 `activate()`** void smtrat::cad::Origin::BaseType::activate ( const carl::Bitset & *ids* ) [inline]

**0.15.33.2.2 `active()`** bool smtrat::cad::Origin::BaseType::active ( ) const [inline]

**0.15.33.2.3 `deactivate()`** void smtrat::cad::Origin::BaseType::deactivate ( const carl::Bitset & *ids* ) [inline]

**0.15.33.2.4 `operator<()`** bool smtrat::cad::Origin::BaseType::operator< ( const BaseType & *bt* ) const [inline]

**0.15.33.2.5 `operator==()`** bool smtrat::cad::Origin::BaseType::operator== ( const BaseType & *bt* ) const [inline]

### 0.15.33.3 Friends And Related Function Documentation

**0.15.33.3.1 `operator<<`** std::ostream& operator<< ( std::ostream & *os*, const BaseType & *bt* ) [friend]

### 0.15.33.4 Field Documentation

**0.15.33.4.1 `ec_active`** bool smtrat::cad::Origin::BaseType::ec\_active = true

**0.15.33.4.2 `first`** std::size\_t smtrat::cad::Origin::BaseType::first

**0.15.33.4.3 `first_active`** bool smtrat::cad::Origin::BaseType::first\_active = true

**0.15.33.4.4 `level`** std::size\_t smtrat::cad::Origin::BaseType::level

**0.15.33.4.5 `second`** std::size\_t smtrat::cad::Origin::BaseType::second

**0.15.33.4.6 `second_active`** bool smtrat::cad::Origin::BaseType::second\_active = true

### 0.15.34 smtrat::mcsat::onecell::BCApproximationSettings Struct Reference

```
#include <onecell.h>
```

#### Static Public Attributes

- constexpr static auto `cell_apx_heuristic` = `cadcells::representation::BIGGEST_CELL_APPROXIMATION`
- constexpr static auto `cell_heuristic` = `cadcells::representation::BIGGEST_CELL`
- constexpr static auto `covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING`
- constexpr static auto `op` = `cadcells::operators::op::mccallum`

#### 0.15.34.1 Field Documentation

**0.15.34.1.1 `cell_apx_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCApproximationSettings::cell_apx_heuristic` = `cadcells::representation::BIGGEST_CELL_APPROXIMATION` [static], [constexpr]

**0.15.34.1.2 `cell_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCSettings::cell_heuristic` = `cadcells::representation::BIGGEST_CELL` [static], [constexpr], [inherited]

**0.15.34.1.3 `covering_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCSettings::covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING` [static], [constexpr], [inherited]

**0.15.34.1.4 `op`** constexpr static auto `smtrat::mcsat::onecell::BCSettings::op` = `cadcells::operators::op::mccallum` [static], [constexpr], [inherited]

### 0.15.35 smtrat::mcsat::onecell::BCFilteredAllSelectiveSettings Struct Reference

```
#include <onecell.h>
```

#### Static Public Attributes

- constexpr static auto `op` = `cadcells::operators::op::mccallum_filtered_all_selective`
- constexpr static auto `cell_heuristic` = `cadcells::representation::BIGGEST_CELL_EW`
- constexpr static auto `covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING_EW`

#### 0.15.35.1 Field Documentation

**0.15.35.1.1 `cell_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredSettings::cell_heuristic` = `cadcells::representation::BIGGEST_CELL_EW` [static], [constexpr], [inherited]

**0.15.35.1.2 `covering_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredSettings::covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING_EW` [static], [constexpr], [inherited]

**0.15.35.1.3 `op`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredAllSelectiveSettings::op` = `cadcells::operators::op::mccallum_filtered_all_selective` [static], [constexpr]

## 0.15.36 smtrat::mcsat::onecell::BCFilteredAllSettings Struct Reference

```
#include <onecell.h>
```

### Static Public Attributes

- constexpr static auto **op** = cadcells::operators::op::mccallum\_filtered\_all
- constexpr static auto **cell\_heuristic** = cadcells::representation::BIGGEST\_CELL\_EW
- constexpr static auto **covering\_heuristic** = cadcells::representation::BIGGEST\_CELL\_COVERING\_EW

### 0.15.36.1 Field Documentation

**0.15.36.1.1 cell\_heuristic** constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings::cell\_heuristic = cadcells::representation::BIGGEST\_CELL\_EW [static], [constexpr], [inherited]

**0.15.36.1.2 covering\_heuristic** constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings::covering\_heuristic = cadcells::representation::BIGGEST\_CELL\_COVERING\_EW [static], [constexpr], [inherited]

**0.15.36.1.3 op** constexpr static auto smtrat::mcsat::onecell::BCFilteredAllSettings::op = cadcells::operators::op::mccallum\_filtered\_all [static], [constexpr]

## 0.15.37 smtrat::mcsat::onecell::BCFilteredBoundsSettings Struct Reference

```
#include <onecell.h>
```

### Static Public Attributes

- constexpr static auto **op** = cadcells::operators::op::mccallum\_filtered\_bounds
- constexpr static auto **cell\_heuristic** = cadcells::representation::BIGGEST\_CELL\_EW
- constexpr static auto **covering\_heuristic** = cadcells::representation::BIGGEST\_CELL\_COVERING\_EW

### 0.15.37.1 Field Documentation

**0.15.37.1.1 cell\_heuristic** constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings::cell\_heuristic = cadcells::representation::BIGGEST\_CELL\_EW [static], [constexpr], [inherited]

**0.15.37.1.2 covering\_heuristic** constexpr static auto smtrat::mcsat::onecell::BCFilteredSettings::covering\_heuristic = cadcells::representation::BIGGEST\_CELL\_COVERING\_EW [static], [constexpr], [inherited]

**0.15.37.1.3 op** constexpr static auto smtrat::mcsat::onecell::BCFilteredBoundsSettings::op = cadcells::operators::op::mccallum\_filtered\_bounds [static], [constexpr]

## 0.15.38 smtrat::mcsat::onecell::BCFilteredSamplesSettings Struct Reference

```
#include <onecell.h>
```

**Static Public Attributes**

- constexpr static auto `op` = `cadcells::operators::op::mccallum_filtered_samples`
- constexpr static auto `cell_heuristic` = `cadcells::representation::BIGGEST_CELL_EW`
- constexpr static auto `covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING_EW`

**0.15.38.1 Field Documentation**

**0.15.38.1.1 `cell_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredSettings::cell_heuristic` = `cadcells::representation::BIGGEST_CELL_EW` [static], [constexpr], [inherited]

**0.15.38.1.2 `covering_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredSettings::covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING_EW` [static], [constexpr], [inherited]

**0.15.38.1.3 `op`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredSamplesSettings::op` = `cadcells::operators::op::mccallum_filtered_samples` [static], [constexpr]

**0.15.39 smtrat::mcsat::onecell::BCFilteredSettings Struct Reference**

```
#include <onecell.h>
```

**Static Public Attributes**

- constexpr static auto `cell_heuristic` = `cadcells::representation::BIGGEST_CELL_EW`
- constexpr static auto `covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING_EW`
- constexpr static auto `op` = `cadcells::operators::op::mccallum_filtered`

**0.15.39.1 Field Documentation**

**0.15.39.1.1 `cell_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredSettings::cell_heuristic` = `cadcells::representation::BIGGEST_CELL_EW` [static], [constexpr]

**0.15.39.1.2 `covering_heuristic`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredSettings::covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING_EW` [static], [constexpr]

**0.15.39.1.3 `op`** constexpr static auto `smtrat::mcsat::onecell::BCFilteredSettings::op` = `cadcells::operators::op::mccallum_filtered` [static], [constexpr]

**0.15.40 smtrat::mcsat::onecell::BCSettings Struct Reference**

```
#include <onecell.h>
```

**Static Public Attributes**

- constexpr static auto `cell_heuristic` = `cadcells::representation::BIGGEST_CELL`
- constexpr static auto `covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING`
- constexpr static auto `op` = `cadcells::operators::op::mccallum`

### 0.15.40.1 Field Documentation

**0.15.40.1.1 `cell_heuristic`** `constexpr static auto smtrat::mcsat::onecell::BCSettings::cell_←heuristic = cadcells::representation::BIGGEST_CELL [static], [constexpr]`

**0.15.40.1.2 `covering_heuristic`** `constexpr static auto smtrat::mcsat::onecell::BCSettings::covering←_heuristic = cadcells::representation::BIGGEST_CELL_COVERING [static], [constexpr]`

**0.15.40.1.3 `op`** `constexpr static auto smtrat::mcsat::onecell::BCSettings::op = cadcells::operators←::op::mccallum [static], [constexpr]`

## 0.15.41 `smtrat::BEModule< Settings >` Class Template Reference

```
#include <BEModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
- `BEModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~BEModule ()`
- `Answer checkCore ()`

*Checks the received formula for consistency.*
- `bool isPreprocessor () const`
- `bool appliedPreprocessing () const`
- `bool add (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- `void remove (ModuleInput::const_iterator _subformula)`

*Removes everything related to the given sub-formula of the received formula.*
- `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`

*Checks the received formula for consistency.*
- `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*
- `virtual Answer runBackends ()`
- `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void updateModel () const`

*Updates the current assignment into the model.*
- `bool inform (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- `void deinform (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- `virtual void init ()`

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `virtual void updateAllModels ()`

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`

*Partition the variables from the current model into equivalence classes according to their assigned value.*

- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`

*Sets this modules unique ID to identify itself.*

- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*

- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`
- `const std::vector< Model > & allModels () const`
- `const std::vector< FormulaSetT > & infeasibleSubsets () const`
- `const std::vector< Module * > & usedBackends () const`
- `const carl::FastSet< FormulaT > & constraintsToInform () const`
- `const carl::FastSet< FormulaT > & informedConstraints () const`
- `void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*

- `bool hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*

- `void clearLemmas ()`

*Deletes all yet found lemmas.*

- `const std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- `void receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*

- `const smrat::Conditionals & answerFound () const`
- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*

- `void updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*

- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*

- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*

- `void print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*

- `void printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*

- void `printPassedFormula` (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const  
*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< `Branching` > `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< `FormulaT` > `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)  
*The inverse of informing about a constraint.*
- virtual bool `addCore` (`ModuleInput`::const\_iterator formula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (`ModuleInput`::const\_iterator formula)  
*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput`::iterator `passedFormulaBegin` ()
- `ModuleInput`::iterator `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput`::iterator \_formula, const `FormulaT` &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput`::const\_iterator \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const

- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
  - std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
    - std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
  - void `informBackends` (const `FormulaT` &\_constraint)
    - Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
  - Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
  - Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
    - Adds the given formula to the passed formula with no origin.*
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
    - Adds the given formula to the passed formula.*
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
    - Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
  - Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
  - Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
  - void `getInfeasibleSubsets` ()
    - Copies the infeasible subsets of the passed formula.*
  - std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
    - Get the infeasible subsets the given backend provides.*
  - const `Model` & `backendsModel` () const
    - Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - void `getBackendsModel` () const
    - Stores all models of a backend in the list of all models of this module.*
  - void `getBackendsAllModels` () const
    - Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
  - void `clearPassedFormula` ()
    - Merges the two vectors of sets into the first one.*
  - size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
    - bool `probablyLooping` (const typename `Poly::PolyType` &\_branchingPolynomial, const `Rational` &\_branchingValue) const
      - Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
    - bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
      - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*

- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector<`FormulaT`> &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set<`FormulaT`> & `getInformationRelevantFormulas` ()
 

*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)
 

*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
 

*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector<`FormulaSetT`> `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector<`Model`> `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector<`TheoryPropagation`> `mTheoryPropagations`
- std::atomic<`Answer`> `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals` `mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector<`Module` \*> `mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector<`Module` \*> `mAllBackends`

*The backends of this module which have been used.*

- `std::vector< Lemma > mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- `carl::FastSet< FormulaT > mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

#### 0.15.41.1 Member Typedef Documentation

**0.15.41.1.1 SettingsType** `template<typename Settings >`  
`typedef Settings smtrat::BEModule< Settings >::SettingsType`

#### 0.15.41.2 Member Enumeration Documentation

**0.15.41.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

#### 0.15.41.3 Constructor & Destructor Documentation

**0.15.41.3.1 BEModule()** `template<typename Settings >`  
`smtrat::BEModule< Settings >::BEModule (`  
 `const ModuleInput * _formula,`  
 `Conditionals & _conditionals,`  
 `Manager * _manager = NULL )`

**0.15.41.3.2 ~BEModule()** `template<typename Settings >`  
`smtrat::BEModule< Settings >::~BEModule ( )`

#### 0.15.41.4 Member Function Documentation

**0.15.41.4.1 add()** `bool smtrat::PModule::add (`  
 `ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

#### **0.15.41.4.2 `addConstraintToInform()`** `void smtrat::Module::addConstraintToInform ( const FormulaT & _constraint )` [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

#### **0.15.41.4.3 `addCore()`** `virtual bool smtrat::Module::addCore ( ModuleInput::const_iterator formula )` [inline], [protected], [virtual], [inherited]

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICEModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::GBModule< Settings >](#).

#### **0.15.41.4.4 `addInformationRelevantFormula()`** `void smtrat::Module::addInformationRelevantFormula ( const FormulaT & formula )` [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

---

**0.15.41.4.5 addLemma()** void smtrat::Module::addLemma (

```
const FormulaT & _lemma,
const LemmaType & _lt = LemmaType::NORMAL,
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| _lemma            | The eduction/lemma to store.                                          |
| _lt               | The type of the lemma.                                                |
| _preferredFormula | Hint for the next decision, which formula should be assigned to true. |

**0.15.41.4.6 addOrigin()** void smtrat::Module::addOrigin (

```
ModuleInput::iterator _formula,
const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

**0.15.41.4.7 addReceivedSubformulaToPassedFormula()** std::pair<ModuleInput::iterator,bool>

```
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

**0.15.41.4.8 addSubformulaToPassedFormula()** [1/3] std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (

```
const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.41.4.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat→
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                 |
| <i>_origin</i>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.41.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool> smtrat→
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector<FormulaT> >& _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.41.4.11 allModels() const std::vector<Model>& smtrat::Module::allModels () const [inline],
[inherited]
```

#### Returns

All satisfying assignments, if existent.

```
0.15.41.4.12 anAnswerFound() bool smtrat::Module::anAnswerFound () const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

**0.15.41.4.13 answerFound()** const `smtrat::Conditionals`& `smtrat::Module::answerFound` ( ) const [inline], [inherited]

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.41.4.14 appliedPreprocessing()** bool `smtrat::PModule::appliedPreprocessing` ( ) const [inline], [inherited]

**Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.41.4.15 backendsModel()** const `Model` & `smtrat::Module::backendsModel` ( ) const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.41.4.16 branchAt() [1/4]** bool `smtrat::Module::branchAt` (

```
 carl::Variable::Arg _var,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.41.4.17 branchAt() [2/4]** bool `smtrat::Module::branchAt` (

```
 carl::Variable::Arg _var,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.41.4.18 branchAt() [3/4]** bool `smtrat::Module::branchAt` (

```
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.41.4.19 branchAt() [4/4]** bool `smtrat::Module::branchAt` (

```
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
```

```

 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]

```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

#### 0.15.41.4.20 `check()`

```

 Answer smtrat::PModule::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]

```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

#### 0.15.41.4.21 `checkCore()`

```

 template<typename Settings >
 Answer smtrat::BEModule< Settings >::checkCore () [virtual]

```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.41.4.22 checkInfSubsetForMinimality() void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.41.4.23 checkModel() unsigned smtrat::Module::checkModel () const [protected], [inherited]
```

#### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

```
0.15.41.4.24 cleanModel() void smtrat::Module::cleanModel () const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

```
0.15.41.4.25 clearLemmas() void smtrat::Module::clearLemmas () [inline], [inherited]
```

Deletes all yet found lemmas.

```
0.15.41.4.26 clearModel() void smtrat::Module::clearModel () const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

```
0.15.41.4.27 clearModels() void smtrat::Module::clearModels () const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

```
0.15.41.4.28 clearPassedFormula() void smtrat::Module::clearPassedFormula () [protected], [inherited]
```

```
0.15.41.4.29 collectOrigins() [1/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

```
0.15.41.4.30 collectOrigins() [2/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                 |                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <i>_origins</i> | The set in which to store the origins.                                                               |

```
0.15.41.4.31 collectTheoryPropagations() void smtrat::Module::collectTheoryPropagations ()
[inherited]
```

```
0.15.41.4.32 constraintsToInform() const carl::FastSet<FormulaT>& smtrat::Module::constraints←
ToInform () const [inline], [inherited]
```

#### Returns

The constraints which the backends must be informed about.

```
0.15.41.4.33 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

```
0.15.41.4.34 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.41.4.35 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                    |                                                         |
|--------------------|---------------------------------------------------------|
| <i>_constraint</i> | The constraint to remove from internal data structures. |
|--------------------|---------------------------------------------------------|

**0.15.41.4.36 `deinformCore()`** `virtual void smtrat::Module::deinformCore (`  
    `const FormulaT & _constraint )` [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.41.4.37 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (`  
    `const std::vector< FormulaT > & origins ) const` [protected], [inherited]

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.41.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`  
    `ModuleInput::iterator _subformula,`  
    `bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.41.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.41.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`  
    `const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

## Parameters

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

## Returns

**0.15.41.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.41.4.42 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

## Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.41.4.43 `freeSplittingVariable()`** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.41.4.44 `generateTrivialInfeasibleSubset()`** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.41.4.45 `getBackendsAllModels()`** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.41.4.46 `getBackendsModel()`** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.41.4.47 `getInfeasibleSubsets()` [1/2]** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.41.4.48 `getInfeasibleSubsets()` [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.41.4.49 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.41.4.50 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.41.4.51 `getOrigins() [1/3]`** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.41.4.52 `getOrigins() [2/3]`** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.41.4.53 `getOrigins() [3/3]`** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.41.4.54 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified( ) [virtual], [inherited]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.41.4.55 `hasLemmas()`** `bool smtrat::Module::hasLemmas( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

**0.15.41.4.56 `isValidInfeasibleSubset()`** `bool smtrat::Module::isValidInfeasibleSubset( ) const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.41.4.57 `id()`** `std::size_t smtrat::Module::id( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.41.4.58 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.41.4.59 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.41.4.60 informBackends()** void smrat::Module::informBackends ( const FormulaT & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.41.4.61 informCore()** virtual bool smrat::Module::informCore (

const FormulaT & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.41.4.62 informedConstraints()** const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.41.4.63 init()** void smrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

**0.15.41.4.64 is\_minimizing()** bool smrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.41.4.65 isLemmaLevel()** `bool smtrat::Module::isLemmaLevel (``LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.41.4.66 isPreprocessor()** `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.41.4.67 lemmas()** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.41.4.68 merge()** `std::vector< FormulaT > smtrat::Module::merge (``const std::vector< FormulaT > & _vecSetA,``const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.41.4.69 model()** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.41.4.70 modelsDisjoint()** `bool smtrat::Module::modelsDisjoint (``const Model & _modelA,``const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.41.4.71 `moduleName()`** template<typename Settings >  
std::string smtrat::BEModule< Settings >::moduleName ( ) const [inline], [virtual]

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.41.4.72 `objective()`** carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]

**0.15.41.4.73 `originInReceivedFormula()`** bool smtrat::Module::originInReceivedFormula ( const FormulaT & \_origin ) const [protected], [inherited]

**0.15.41.4.74 `passedFormulaBegin()`** ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.41.4.75 `passedFormulaEnd()`** ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.41.4.76 `pPassedFormula()`** const ModuleInput\* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.41.4.77 `pReceivedFormula()`** const ModuleInput\* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.41.4.78 `print()`** void smtrat::Module::print ( const std::string & \_initiation = "\*\*\*" ) const [inherited]

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.41.4.79 `printAllModels()`** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.41.4.80 `printInfeasibleSubsets()`** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.41.4.81 `printModel()`** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.41.4.82 `printPassedFormula()`** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.41.4.83 `printReceivedFormula()`** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.41.4.84 `probablyLooping()`** `bool smrat::Module::probablyLooping (`  
    `const typename Poly::PolyType & _branchingPolynomial,`  
    `const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.41.4.85 `receivedFormulaChecked()`** `void smrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.41.4.86 `receivedFormulasAsInfeasibleSubset()`** `void smrat::Module::receivedFormulasAsInfeasibleSubset (`  
    `ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.41.4.87 `receivedVariable()`** `bool smrat::Module::receivedVariable (`  
    `carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.41.4.88 `remove()`** `void smrat::PModule::remove (`  
    `ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.41.4.89 `removeCore()`** `virtual void smrat::Module::removeCore (`  
    `ModuleInput::const_iterator formula ) [inline], [protected], [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.41.4.90 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin(`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.41.4.91 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins(`

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector<FormulaT>> & _origins) [inline], [protected],
[inherited]
```

**0.15.41.4.92 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula( ) const`

[inline], [inherited]

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.41.4.93 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula( ) const`

[inline], [inherited]

#### Returns

A reference to the formula passed to this module.

**0.15.41.4.94 runBackends() [1/2]** `virtual Answer smrat::PModule::runBackends( ) [inline], [virtual], [inherited]`

Reimplemented from [smrat::Module](#).

**0.15.41.4.95 runBackends() [2/2]** `Answer smrat::PModule::runBackends(`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.41.4.96 setId()** `void smrat::Module::setId ( std::size_t _id ) [inline], [inherited]`  
Sets this modules unique ID to identify itself.

**Parameters**

|              |                                    |
|--------------|------------------------------------|
| $\leftarrow$ | The id to set this module's id to. |
| $\_id$       |                                    |

**0.15.41.4.97 setThreadPriority()** `void smrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`  
Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| $\_threadPriority$ | The priority to set this module's thread priority to. |
|--------------------|-------------------------------------------------------|

**0.15.41.4.98 solverState()** `Answer smrat::Module::solverState ( ) const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.41.4.99 splitUnequalConstraint()** `void smrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                       |                                                  |
|-----------------------|--------------------------------------------------|
| $\_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|-----------------------|--------------------------------------------------|

**0.15.41.4.100 threadPriority()** `thread_priority smrat::Module::threadPriority ( ) const [inline], [inherited]`

**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.41.4.101 updateAllModels()** void smrat::Module::updateAllModels () [virtual], [inherited]  
Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in [smrat::SATModule< Settings >](#).

**0.15.41.4.102 updateLemmas()** void smrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.41.4.103 updateModel()** void smrat::PModule::updateModel () const [virtual], [inherited]  
Updates the current assignment into the model.  
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smrat::Module](#).

**0.15.41.4.104 usedBackends()** const std::vector<[Module](#)\*>& smrat::Module::usedBackends ()  
const [inline], [inherited]

Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.41.5 Field Documentation

**0.15.41.5.1 mAllBackends** std::vector<[Module](#)\*> smrat::Module::mAllBackends [protected], [inherited]  
The backends of this module which have been used.

**0.15.41.5.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.41.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.41.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.41.5.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.41.5.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.41.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass` [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.41.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.41.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer` [protected], [inherited]

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.41.5.10 mFullCheck** `bool smrat::Module::mFullCheck` [protected], [inherited]

false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.41.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smrat::Module::mInfeasibleSubsets`

[protected], [inherited]

Stores the infeasible subsets.

**0.15.41.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints`

[protected], [inherited]

Stores the position of the first constraint of which no backend has been informed about.

**0.15.41.5.13 mLastBranches** `std::vector< Branching > smrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.41.5.14 mLemmas** `std::vector<Lemma> smrat::Module::mLemmas` [protected], [inherited]

Stores the lemmas being valid formulas this module or its backends made.

**0.15.41.5.15 mModel** `Model smrat::Module::mModel` [mutable], [protected], [inherited]

Stores the assignment of the current satisfiable result, if existent.

**0.15.41.5.16 mModelComputed** `bool smrat::Module::mModelComputed` [mutable], [protected], [inherited]

True, if the model has already been computed.

**0.15.41.5.17 mNumOfBranchVarsToStore** `std::size_t smrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.41.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
 Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.41.5.19 mOldSplittingVariables** `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]`  
 Reusable splitting variables.

**0.15.41.5.20 mpManager** `Manager* const smtrat::Module::mpManager [protected], [inherited]`  
 A reference to the manager.

**0.15.41.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`  
 Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.41.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`  
 States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.41.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

**0.15.41.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`  
 The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.41.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`  
 Maps variables to the number of their occurrences.

## 0.15.42 benchmax::BenchmarkResult Struct Reference

Results for a single benchmark run.

```
#include <BenchmarkResult.h>
```

### Public Member Functions

- template<typename TimeLimit >  
`void cleanup (const TimeLimit &limit)`  
*Properly detect timeouts.*
- auto `get_path () const`
- void `store (size_t id) const`
- void `store () const`
- void `restore () const`
- void `forget () const`

## Data Fields

- int `exitCode`  
*Shell exit code.*
- std::chrono::milliseconds `time`  
*Runtime in milliseconds.*
- std::string `answer`  
*Answer string.*
- std::size\_t `peak_memory_kbytes`  
*Peak memory usage.*
- std::string `stdout`  
*Standard output (mostly for parsing the answer and additional information).*
- std::string `stderr`  
*Error output (mostly for parsing the answer and additional information).*
- std::map< std::string, std::string > `additional`  
*Arbitrary additional information that can be provided by the tool class.*
- size\_t `stored_id` = 0  
*Identifier for temporary file.*

### 0.15.42.1 Detailed Description

[Results](#) for a single benchmark run.

### 0.15.42.2 Member Function Documentation

**0.15.42.2.1 `cleanup()`** template<typename TimeLimit >  
void benchmax::BenchmarkResult::cleanup (  
 const TimeLimit & limit ) [inline]

Properly detect timeouts.

Most backends give processes a bit more time to avoid having race-condition-like situations.

**0.15.42.2.2 `forget()`** void benchmax::BenchmarkResult::forget ( ) const [inline]

**0.15.42.2.3 `get_path()`** auto benchmax::BenchmarkResult::get\_path ( ) const [inline]

**0.15.42.2.4 `restore()`** void benchmax::BenchmarkResult::restore ( ) const [inline]

**0.15.42.2.5 `store()` [1/2]** void benchmax::BenchmarkResult::store ( ) const [inline]

**0.15.42.2.6 `store()` [2/2]** void benchmax::BenchmarkResult::store (  
 size\_t id ) const [inline]

### 0.15.42.3 Field Documentation

**0.15.42.3.1 `additional`** std::map<std::string, std::string> benchmax::BenchmarkResult::additional  
[mutable]

Arbitrary additional information that can be provided by the tool class.

**0.15.42.3.2 answer** std::string benchmax::BenchmarkResult::answer  
Answer string.

**0.15.42.3.3 exitCode** int benchmax::BenchmarkResult::exitCode  
Shell exit code.

**0.15.42.3.4 peak\_memory\_kbytes** std::size\_t benchmax::BenchmarkResult::peak\_memory\_kbytes  
Peak memory usage.

**0.15.42.3.5 stderr** std::string benchmax::BenchmarkResult::stderr  
Error output (mostly for parsing the answer and additional information).

**0.15.42.3.6 stdout** std::string benchmax::BenchmarkResult::stdout  
Standard output (mostly for parsing the answer and additional information).

**0.15.42.3.7 stored\_id** size\_t benchmax::BenchmarkResult::stored\_id = 0 [mutable]  
Identifier for temporary file.

**0.15.42.3.8 time** std::chrono::milliseconds benchmax::BenchmarkResult::time  
Runtime in milliseconds.

## 0.15.43 benchmax::BenchmarkSet Class Reference

A set of benchmarks from some common base directory.

```
#include <BenchmarkSet.h>
```

### Public Member Functions

- void **add\_directory** (const std::filesystem::path &dir)  
*Recursively find all benchmarks from this directory.*
- std::size\_t **size** () const  
*Number of files.*
- auto **begin** () const  
*Begin iterator.*
- auto **end** () const  
*End iterator.*

### 0.15.43.1 Detailed Description

A set of benchmarks from some common base directory.

### 0.15.43.2 Member Function Documentation

**0.15.43.2.1 add\_directory()** void benchmax::BenchmarkSet::add\_directory (  
    const std::filesystem::path & dir )  
Recursively find all benchmarks from this directory.

**0.15.43.2.2 `begin()`** `auto benchmax::BenchmarkSet::begin() const [inline]`  
Begin iterator.

**0.15.43.2.3 `end()`** `auto benchmax::BenchmarkSet::end() const [inline]`  
End iterator.

**0.15.43.2.4 `size()`** `std::size_t benchmax::BenchmarkSet::size() const [inline]`  
Number of files.

## 0.15.44 `benchmax::settings::BenchmarkSettings` Struct Reference

[Settings](#) for benchmarks.

```
#include <benchmarks.h>
```

### Data Fields

- `carl::settings::binary_quantity limit_memory`  
*Memory limit in megabytes.*
- `carl::settings::duration limit_time`  
*Time limit in seconds.*
- `carl::settings::duration grace_time`  
*Grace time in seconds.*
- `std::vector< std::filesystem::path > input_directories`  
*Lift of input directories.*
- `std::filesystem::path input_directories_common_prefix`  
*Common prefix of input directories (to shorten filenames in output).*
- `std::filesystem::path output_dir`  
*Output directory.*
- `std::filesystem::path output_file_xml`  
*Filename of xml file.*

### 0.15.44.1 Detailed Description

[Settings](#) for benchmarks.

### 0.15.44.2 Field Documentation

**0.15.44.2.1 `grace_time`** `carl::settings::duration benchmax::settings::BenchmarkSettings::grace_time`  
Grace time in seconds.

**0.15.44.2.2 `input_directories`** `std::vector<std::filesystem::path> benchmax::settings::BenchmarkSettings::input_directories`  
Lift of input directories.

**0.15.44.2.3 `input_directories_common_prefix`** `std::filesystem::path benchmax::settings::BenchmarkSettings::input_directories_common_prefix`  
Common prefix of input directories (to shorten filenames in output).

**0.15.44.2.4 limit\_memory** carl::settings::binary\_quantity benchmax::settings::BenchmarkSettings::limit\_memory  
Memory limit in megabytes.

**0.15.44.2.5 limit\_time** carl::settings::duration benchmax::settings::BenchmarkSettings::limit\_time  
Time limit in seconds.

**0.15.44.2.6 output\_dir** std::filesystem::path benchmax::settings::BenchmarkSettings::output\_dir  
Output directory.

**0.15.44.2.7 output\_file\_xml** std::filesystem::path benchmax::settings::BenchmarkSettings::output\_file\_xml  
Filename of xml file.

## 0.15.45 smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName > Class Template Reference

Container that stores expensive to construct objects and allows the fast lookup with respect to two independent keys within the objects.

```
#include <Bimap.h>
```

### Public Types

- `typedef std::forward_list< Class > Data`
- `typedef Data::iterator Iterator`
- `typedef Data::const_iterator ConstIterator`

### Public Member Functions

- `Iterator begin () noexcept`
- `ConstIterator begin () const noexcept`
- `Iterator end () noexcept`
- `ConstIterator end () const noexcept`
- `Class & firstAt (const FirstKeyType &firstKey)`
- `Class & secondAt (const SecondKeyType &secondKey)`
- `Iterator firstFind (const FirstKeyType &firstKey)`
- `Iterator secondFind (const SecondKeyType &secondKey)`
- `template<typename... Args> Iterator emplace (Args &&... args)`

### 0.15.45.1 Detailed Description

```
template<class Class, typename FirstKeyType, FirstKeyType Class::* FirstKeyName, typename SecondKeyType, SecondKeyType Class::* SecondKeyName>
```

```
class smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >
```

Container that stores expensive to construct objects and allows the fast lookup with respect to two independent keys within the objects.

### 0.15.45.2 Member Typedef Documentation

**0.15.45.2.1 ConstIterator** template<class Class , typename FirstKeyType , FirstKeyType Class::\* FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
typedef Data::const\_iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::ConstIterator

**0.15.45.2.2 Data** template<class Class , typename FirstKeyType , FirstKeyType Class::\* First←KeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
typedef std::forward\_list<Class> smtrat::Bimap< Class, FirstKeyType, FirstKeyName, Second←KeyType, SecondKeyName >::Data

**0.15.45.2.3 Iterator** template<class Class , typename FirstKeyType , FirstKeyType Class::\* First←KeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
typedef Data::iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, Second←KeyName >::Iterator

### 0.15.45.3 Member Function Documentation

**0.15.45.3.1 begin() [1/2]** template<class Class , typename FirstKeyType , FirstKeyType Class::\* FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
ConstIterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::begin ( ) const [inline], [noexcept]

**0.15.45.3.2 begin() [2/2]** template<class Class , typename FirstKeyType , FirstKeyType Class::\* FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::begin ( ) [inline], [noexcept]

**0.15.45.3.3 emplace()** template<class Class , typename FirstKeyType , FirstKeyType Class::\* FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
template<typename... Args>  
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::emplace ( Args &&... args ) [inline]

**0.15.45.3.4 end() [1/2]** template<class Class , typename FirstKeyType , FirstKeyType Class::\* FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
ConstIterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::end ( ) const [inline], [noexcept]

**0.15.45.3.5 end() [2/2]** template<class Class , typename FirstKeyType , FirstKeyType Class::\* FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >::end ( ) [inline], [noexcept]

**0.15.45.3.6 firstAt()** template<class Class , typename FirstKeyType , FirstKeyType Class::\* First←KeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>

```
Class& smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >←
::firstAt (
 const FirstKeyType & firstKey) [inline]
```

**0.15.45.3.7 firstFind()** template<class Class , typename FirstKeyType , FirstKeyType Class::\*  
FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >←
::firstFind (
 const FirstKeyType & firstKey ) [inline]

**0.15.45.3.8 secondAt()** template<class Class , typename FirstKeyType , FirstKeyType Class::\*  
FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
Class& smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >←
::secondAt (
 const SecondKeyType & secondKey ) [inline]

**0.15.45.3.9 secondFind()** template<class Class , typename FirstKeyType , FirstKeyType Class::\*  
FirstKeyName, typename SecondKeyType , SecondKeyType Class::\* SecondKeyName>  
Iterator smtrat::Bimap< Class, FirstKeyType, FirstKeyName, SecondKeyType, SecondKeyName >←
::secondFind (
 const SecondKeyType & secondKey ) [inline]

## 0.15.46 smtrat::expression::BinaryExpression Struct Reference

```
#include <ExpressionContent.h>
```

### Public Member Functions

- [BinaryExpression \(BinaryType \\_type, Expression &&\\_lhs, Expression &&\\_rhs\)](#)

### Data Fields

- [BinaryType type](#)
- [Expression lhs](#)
- [Expression rhs](#)

#### 0.15.46.1 Constructor & Destructor Documentation

**0.15.46.1.1 BinaryExpression()** smtrat::expression::BinaryExpression::BinaryExpression (

```
BinaryType _type,
Expression && _lhs,
Expression && _rhs) [inline]
```

#### 0.15.46.2 Field Documentation

**0.15.46.2.1 lhs** [Expression](#) smtrat::expression::BinaryExpression::lhs

**0.15.46.2.2 rhs** [Expression](#) smtrat::expression::BinaryExpression::rhs

**0.15.46.2.3 type** `BinaryType smtrat::expression::BinaryExpression::type`

## 0.15.47 `smtrat::datastructures::BinaryHeap< T >` Class Template Reference

```
#include <BinaryHeap.h>
```

### Public Member Functions

- `BinaryHeap (std::size_t initialSize=100)`
- `BinaryHeap (const std::vector< T > &elements)`
- `bool is_empty () const`
- `std::size_t size () const`
- `void insert (const T &item)`
- `T extractMin ()`
- `void clear ()`

#### 0.15.47.1 Constructor & Destructor Documentation

**0.15.47.1.1 `BinaryHeap()` [1/2]** `template<typename T >`  
`smtrat::datastructures::BinaryHeap< T >::BinaryHeap (`  
 `std::size_t initialSize = 100 ) [inline], [explicit]`

**0.15.47.1.2 `BinaryHeap()` [2/2]** `template<typename T >`  
`smtrat::datastructures::BinaryHeap< T >::BinaryHeap (`  
 `const std::vector< T > & elements ) [inline], [explicit]`

#### 0.15.47.2 Member Function Documentation

**0.15.47.2.1 `clear()`** `template<typename T >`  
`void smtrat::datastructures::BinaryHeap< T >::clear () [inline]`

**0.15.47.2.2 `extractMin()`** `template<typename T >`  
`T smtrat::datastructures::BinaryHeap< T >::extractMin () [inline]`

**0.15.47.2.3 `insert()`** `template<typename T >`  
`void smtrat::datastructures::BinaryHeap< T >::insert (`  
 `const T & item ) [inline]`

**0.15.47.2.4 `is_empty()`** `template<typename T >`  
`bool smtrat::datastructures::BinaryHeap< T >::is_empty () const [inline]`

**0.15.47.2.5 `size()`** `template<typename T >`  
`std::size_t smtrat::datastructures::BinaryHeap< T >::size () const [inline]`

## 0.15.48 `smtrat::parser::BinaryParser` Struct Reference

Parses binaries: #b[01]+  
`#include <Lexicon.h>`

## Public Types

- `typedef boost::iterator_range< Iterator > ITRange`

## Public Member Functions

- `BinaryParser ()`
- `FixedWidthConstant< Integer > build (const ITRange &itr, const Integer &val)`

## Data Fields

- `qi::uint_parser< Integer, 2, 1,-1 > number`
- `qi::rule< Iterator, FixedWidthConstant< Integer >, Skipper, qi::locals< Integer > > main`
- `qi::rule< Iterator, FixedWidthConstant< Integer >, Skipper > main2`

### 0.15.48.1 Detailed Description

Parses binaries: #b[01] +

### 0.15.48.2 Member Typedef Documentation

#### 0.15.48.2.1 ITRange `typedef boost::iterator_range<Iterator> smtrat::parser::BinaryParser::ITRange`

#### 0.15.48.3 Constructor & Destructor Documentation

#### 0.15.48.3.1 BinaryParser() `smtrat::parser::BinaryParser::BinaryParser ( ) [inline]`

#### 0.15.48.4 Member Function Documentation

#### 0.15.48.4.1 build() `FixedWidthConstant<Integer> smtrat::parser::BinaryParser::build ( const ITRange & itr, const Integer & val ) [inline]`

#### 0.15.48.5 Field Documentation

#### 0.15.48.5.1 main `qi::rule<Iterator, FixedWidthConstant<Integer>, Skipper, qi::locals<Integer> > smtrat::parser::BinaryParser::main`

#### 0.15.48.5.2 main2 `qi::rule<Iterator, FixedWidthConstant<Integer>, Skipper> smtrat::parser::BinaryParser::main2`

#### 0.15.48.5.3 number `qi::uint_parser<Integer,2,1,-1> smtrat::parser::BinaryParser::number`

## 0.15.49 smtrat::parser::BitvectorTheory Struct Reference

Implements the theory of bitvectors.

```
#include <Bitvector.h>
```

## Public Types

- using `OperatorType` = `carl::BVTermType`

## Public Member Functions

- `BitvectorTheory (ParserState *state)`
- `bool declareVariable (const std::string &name, const carl::Sort &sort, types::VariableType &result, TheoryError &errors)`  
*Declare a new variable with the given name and the given sort.*
- `bool resolveSymbol (const Identifier &identifier, types::TermType &result, TheoryError &errors)`  
*Resolve a symbol that was not declared within the `ParserState`.*
- `bool handleITE (const FormulaT &ifterm, const types::TermType &thenterm, const types::TermType &elseterm, types::TermType &result, TheoryError &errors)`  
*Resolve an if-then-else operator.*
- `bool handleDistinct (const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)`  
*Resolve a distinct operator.*
- `bool instantiate (const types::VariableType &var, const types::TermType &replacement, types::TermType &subject, TheoryError &errors)`  
*Instantiate a variable within a term.*
- `bool refreshVariable (const types::VariableType &var, types::VariableType &subject, TheoryError &errors)`
- `bool functionCall (const Identifier &identifier, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)`  
*Resolve another unknown function call.*
- template<typename T, typename Builder>  
`FormulaT expandDistinct (const std::vector< T > &values, const Builder &neqBuilder)`

## Static Public Member Functions

- static void `addSimpleSorts (qi::symbols< char, carl::Sort > &sorts)`
- static void `addConstants (qi::symbols< char, types::ConstType > &)`  
*Initialize the global symbol table for constants.*

## Data Fields

- `carl::Sort bvSort`
- `conversion::VectorVariantConverter< types::BVTerm > vectorConverter`
- `conversion::VariantConverter< types::BVTerm > termConverter`
- `ParserState * state`

### 0.15.49.1 Detailed Description

Implements the theory of bitvectors.

### 0.15.49.2 Member Typedef Documentation

#### 0.15.49.2.1 OperatorType using `smtrat::parser::BitvectorTheory::OperatorType` = `carl::BVTermType`

### 0.15.49.3 Constructor & Destructor Documentation

**0.15.49.3.1 BitvectorTheory()** smtrat::parser::BitvectorTheory::BitvectorTheory ( ParserState \* state )

#### 0.15.49.4 Member Function Documentation

**0.15.49.4.1 addConstants()** static void smtrat::parser::AbstractTheory::addConstants ( qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]  
Initialize the global symbol table for constants.

**0.15.49.4.2 addSimpleSorts()** void smtrat::parser::BitvectorTheory::addSimpleSorts ( qi::symbols< char, carl::Sort > & sorts ) [static]

**0.15.49.4.3 declareVariable()** bool smtrat::parser::BitvectorTheory::declareVariable ( const std::string & , const carl::Sort & , types::VariableType & , TheoryError & errors ) [virtual]

Declare a new variable with the given name and the given sort.

Reimplemented from [smtrat::parser::AbstractTheory](#).

**0.15.49.4.4 expandDistinct()** template<typename T , typename Builder > FormulaT smtrat::parser::AbstractTheory::expandDistinct ( const std::vector< T > & values, const Builder & neqBuilder ) [inline], [inherited]

**0.15.49.4.5 functionCall()** bool smtrat::parser::BitvectorTheory::functionCall ( const Identifier & , const std::vector< types::TermType > & , types::TermType & , TheoryError & errors ) [virtual]

Resolve another unknown function call.

Reimplemented from [smtrat::parser::AbstractTheory](#).

**0.15.49.4.6 handleDistinct()** bool smtrat::parser::BitvectorTheory::handleDistinct ( const std::vector< types::TermType > & , types::TermType & , TheoryError & errors ) [virtual]

Resolve a distinct operator.

Reimplemented from [smtrat::parser::AbstractTheory](#).

**0.15.49.4.7 handleITE()** bool smtrat::parser::BitvectorTheory::handleITE ( const FormulaT & , const types::TermType & , const types::TermType & , types::TermType & , TheoryError & errors ) [virtual]

Resolve an if-then-else operator.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.49.4.8 instantiate() bool smtrat::parser::BitvectorTheory::instantiate (
 const types::VariableType & ,
 const types::TermType & ,
 types::TermType & ,
 TheoryError & errors) [virtual]
```

Instantiate a variable within a term.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.49.4.9 refreshVariable() bool smtrat::parser::BitvectorTheory::refreshVariable (
 const types::VariableType & var,
 types::VariableType & subject,
 TheoryError & errors) [virtual]
```

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.49.4.10 resolveSymbol() bool smtrat::parser::BitvectorTheory::resolveSymbol (
 const Identifier & ,
 types::TermType & ,
 TheoryError & errors) [virtual]
```

Resolve a symbol that was not declared within the [ParserState](#).

This might be a symbol that actually uses indices, for example bitvector constants.

Reimplemented from [smtrat::parser::AbstractTheory](#).

## 0.15.49.5 Field Documentation

**0.15.49.5.1 bvSort** carl::Sort smtrat::parser::BitvectorTheory::bvSort

**0.15.49.5.2 state** ParserState\* smtrat::parser::AbstractTheory::state [inherited]

**0.15.49.5.3 termConverter** conversion::VariantConverter<types::BVTerm> smtrat::parser::BitvectorTheory::termConverter

**0.15.49.5.4 vectorConverter** conversion::VectorVariantConverter<types::BVTerm> smtrat::parser::BitvectorTheory::vectorConverter

## 0.15.50 smtrat::parser::types::BitvectorTheory Struct Reference

Types of the theory of bitvectors.

```
#include <TheoryTypes.h>
```

### Public Types

- `typedef mpl::vector< BVVariable, FixedWidthConstant< Integer >, BVTerm > ConstTypes`
- `typedef mpl::vector< BVVariable > VariableTypes`
- `typedef mpl::vector< BVVariable, FixedWidthConstant< Integer >, BVTerm, BVConstraint > ExpressionTypes`
- `typedef mpl::vector< BVVariable, FixedWidthConstant< Integer >, BVTerm, BVConstraint > TermTypes`
- `typedef carl::mpl_variant_of< TermTypes >::type TermType`

### 0.15.50.1 Detailed Description

Types of the theory of bitvectors.

### 0.15.50.2 Member Typedef Documentation

**0.15.50.2.1 ConstTypes** `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm> smtrat::parser::types::BitvectorTheory::ConstTypes`

**0.15.50.2.2 ExpressionTypes** `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm, BVConstraint> smtrat::parser::types::BitvectorTheory::ExpressionTypes`

**0.15.50.2.3 TermType** `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::BitvectorTheory::TermType`

**0.15.50.2.4 TermTypes** `typedef mpl::vector<BVVariable, FixedWidthConstant<Integer>, BVTerm, BVConstraint> smtrat::parser::types::BitvectorTheory::TermTypes`

**0.15.50.2.5 VariableTypes** `typedef mpl::vector<BVVariable> smtrat::parser::types::BitvectorTheory::VariableTypes`

### 0.15.51 smtrat::BlastedConstr Class Reference

```
#include <IntBlastModule.h>
```

#### Public Member Functions

- `BlastedConstr()`
- `BlastedConstr(const FormulaT &_formula, const FormulasT &_constraints)`
- `BlastedConstr(const FormulaT &_formula)`
- `BlastedConstr(bool _satisfied)`
- `const FormulaT & formula() const`
- `const FormulasT & constraints() const`

#### Friends

- `std::ostream & operator<< (std::ostream &_out, const BlastedConstr &_constr)`

#### 0.15.51.1 Constructor & Destructor Documentation

**0.15.51.1.1 BlastedConstr() [1/4]** `smtrat::BlastedConstr::BlastedConstr() [inline]`

**0.15.51.1.2 BlastedConstr() [2/4]** `smtrat::BlastedConstr::BlastedConstr(`  
`const FormulaT & _formula,`  
`const FormulasT & _constraints) [inline]`

**0.15.51.1.3 BlastedConstr() [3/4]** `smtrat::BlastedConstr::BlastedConstr(`  
`const FormulaT & _formula) [inline]`

**0.15.51.1.4 BlastedConstr() [4/4]** `smtrat::BlastedConstr::BlastedConstr(`  
`bool _satisfied) [inline]`

### 0.15.51.2 Member Function Documentation

**0.15.51.2.1 constraints()** const `FormulasT&` `smtrat::BlastedConstr::constraints () const` [inline]

**0.15.51.2.2 formula()** const `FormulaT&` `smtrat::BlastedConstr::formula () const` [inline]

### 0.15.51.3 Friends And Related Function Documentation

**0.15.51.3.1 operator<<** `std::ostream& operator<< (`  
    `std::ostream & _out,`  
    `const BlastedConstr & _constr )` [friend]

## 0.15.52 smtrat::BlastedPoly Class Reference

#include <IntBlastModule.h>

### Public Member Functions

- `BlastedPoly ()`
- `BlastedPoly (Integer _constant)`
- `BlastedPoly (Integer _constant, FormulasT _constraints)`
- `BlastedPoly (AnnotatedBVTerm _term)`
- `BlastedPoly (AnnotatedBVTerm _term, FormulasT _constraints)`
- `bool is_constant () const`
- `const Integer & constant () const`
- `const AnnotatedBVTerm & term () const`
- `const FormulasT & constraints () const`
- `const Integer & lower_bound () const`
- `const Integer & upper_bound () const`

### Friends

- `std::ostream & operator<< (std::ostream & _out, const BlastedPoly & _poly)`

### 0.15.52.1 Constructor & Destructor Documentation

**0.15.52.1.1 BlastedPoly() [1/5]** `smtrat::BlastedPoly::BlastedPoly ()` [inline]

**0.15.52.1.2 BlastedPoly() [2/5]** `smtrat::BlastedPoly::BlastedPoly (`  
    `Integer _constant )` [inline]

**0.15.52.1.3 BlastedPoly() [3/5]** `smtrat::BlastedPoly::BlastedPoly (`  
    `Integer _constant,`  
    `FormulasT _constraints )` [inline]

**0.15.52.1.4 BlastedPoly() [4/5]** smtrat::BlastedPoly::BlastedPoly ( AnnotatedBVTerm \_term ) [inline]

**0.15.52.1.5 BlastedPoly() [5/5]** smtrat::BlastedPoly::BlastedPoly ( AnnotatedBVTerm \_term, FormulasT \_constraints ) [inline]

## 0.15.52.2 Member Function Documentation

**0.15.52.2.1 constant()** const Integer& smtrat::BlastedPoly::constant ( ) const [inline]

**0.15.52.2.2 constraints()** const FormulasT& smtrat::BlastedPoly::constraints ( ) const [inline]

**0.15.52.2.3 is\_constant()** bool smtrat::BlastedPoly::is\_constant ( ) const [inline]

**0.15.52.2.4 lower\_bound()** const Integer& smtrat::BlastedPoly::lower\_bound ( ) const [inline]

**0.15.52.2.5 term()** const AnnotatedBVTerm& smtrat::BlastedPoly::term ( ) const [inline]

**0.15.52.2.6 upper\_bound()** const Integer& smtrat::BlastedPoly::upper\_bound ( ) const [inline]

## 0.15.52.3 Friends And Related Function Documentation

**0.15.52.3.1 operator<<** std::ostream& operator<< ( std::ostream & \_out, const BlastedPoly & \_poly ) [friend]

## 0.15.53 smtrat::mcsat::Bookkeeping Class Reference

Represent the trail, i.e.

```
#include <Bookkeeping.h>
```

### Public Member Functions

- const auto & **model** () const
- const auto & **assignedVariables** () const
- const auto & **assignments** () const
- const auto & **constraints** () const
- const auto & **mvBounds** () const
- const auto & **variables** () const
- void **updateVariables** (const carl::Variables &**variables**)
- void **pushConstraint** (const FormulaT &f)
 

*Assert a constraint/literal.*
- void **popConstraint** (const FormulaT &f)
- void **pushAssignment** (carl::Variable v, const ModelValue &mv, const FormulaT &f)
- void **popAssignment** (carl::Variable v)

### 0.15.53.1 Detailed Description

Represent the trail, i.e.

the assignment/model state, of a MCSAT run in different representations (kept in sync) for a fast access. Most notably, we store literals, i.e. a polynomial (in)equality-atom or its negation, which we assert to be true. Since the negation can be represented by an atom by flipping the (in)equality, it suffices to store only atoms. Here we call atomic formulas over the theory of real arithmetic "constraints".

### 0.15.53.2 Member Function Documentation

**0.15.53.2.1 assignedVariables()** const auto& smtrat::mcsat::Bookkeeping::assignedVariables ( )  
const [inline]

**0.15.53.2.2 assignments()** const auto& smtrat::mcsat::Bookkeeping::assignments ( ) const [inline]

**0.15.53.2.3 constraints()** const auto& smtrat::mcsat::Bookkeeping::constraints ( ) const [inline]

**0.15.53.2.4 model()** const auto& smtrat::mcsat::Bookkeeping::model ( ) const [inline]

**0.15.53.2.5 mvBounds()** const auto& smtrat::mcsat::Bookkeeping::mvBounds ( ) const [inline]

**0.15.53.2.6 popAssignment()** void smtrat::mcsat::Bookkeeping::popAssignment ( carl::Variable v ) [inline]

**0.15.53.2.7 popConstraint()** void smtrat::mcsat::Bookkeeping::popConstraint ( const FormulaT & f ) [inline]

**0.15.53.2.8 pushAssignment()** void smtrat::mcsat::Bookkeeping::pushAssignment ( carl::Variable v, const ModelValue & mv, const FormulaT & f ) [inline]

**0.15.53.2.9 pushConstraint()** void smtrat::mcsat::Bookkeeping::pushConstraint ( const FormulaT & f ) [inline]

Assert a constraint/literal.

**0.15.53.2.10 updateVariables()** void smtrat::mcsat::Bookkeeping::updateVariables ( const carl::Variables & variables ) [inline]

**0.15.53.2.11 variables()** const auto& smtrat::mcsat::Bookkeeping::variables ( ) const [inline]

## 0.15.54 smtrat::parser::BooleanEncodingTheory Struct Reference

Implements the theory of bitvectors.

```
#include <BooleanEncoding.h>
```

## Public Member Functions

- `BooleanEncodingTheory (ParserState *state)`
- virtual bool `declareVariable (const std::string &, const carl::Sort &, types::VariableType &, TheoryError &errors)`  
*Declare a new variable with the given name and the given sort.*
- virtual bool `resolveSymbol (const Identifier &, types::TermType &, TheoryError &errors)`  
*Resolve a symbol that was not declared within the ParserState.*
- virtual bool `handleITE (const FormulaT &, const types::TermType &, const types::TermType &, types::TermType &, TheoryError &errors)`  
*Resolve an if-then-else operator.*
- virtual bool `handleDistinct (const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)`  
*Resolve a distinct operator.*
- template<typename T, typename Builder>  
`FormulaT expandDistinct (const std::vector< T > &values, const Builder &neqBuilder)`
- virtual bool `instantiate (const types::VariableType &, const types::TermType &, types::TermType &, TheoryError &errors)`  
*Instantiate a variable within a term.*
- virtual bool `refreshVariable (const types::VariableType &, types::VariableType &, TheoryError &errors)`
- virtual bool `functionCall (const Identifier &, const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)`  
*Resolve another unknown function call.*

## Static Public Member Functions

- static void `addSimpleSorts (qi::symbols< char, carl::Sort > &)`  
*Initialize the global symbol table for simple sorts.*
- static void `addConstants (qi::symbols< char, types::ConstType > &)`  
*Initialize the global symbol table for constants.*

## Data Fields

- `ParserState * state`

### 0.15.54.1 Detailed Description

Implements the theory of bitvectors.

### 0.15.54.2 Constructor & Destructor Documentation

```
0.15.54.2.1 BooleanEncodingTheory() smtrat::parser::BooleanEncodingTheory::BooleanEncoding<→
Theory (
 ParserState * state)
```

### 0.15.54.3 Member Function Documentation

```
0.15.54.3.1 addConstants() static void smtrat::parser::AbstractTheory::addConstants (
 qi::symbols< char, types::ConstType > &) [inline], [static], [inherited]
Initialize the global symbol table for constants.
```

**0.15.54.3.2 addSimpleSorts()** static void smtrat::parser::AbstractTheory::addSimpleSorts ( qi::symbols< char, carl::Sort > & ) [inline], [static], [inherited]  
Initialize the global symbol table for simple sorts.

**0.15.54.3.3 declareVariable()** virtual bool smtrat::parser::AbstractTheory::declareVariable ( const std::string & , const carl::Sort & , types::VariableType & , TheoryError & errors ) [inline], [virtual], [inherited]

Declare a new variable with the given name and the given sort.

Reimplemented in [smtrat::parser::UninterpretedTheory](#), [smtrat::parser::CoreTheory](#), [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

**0.15.54.3.4 expandDistinct()** template<typename T , typename Builder > FormulaT smtrat::parser::AbstractTheory::expandDistinct ( const std::vector< T > & values, const Builder & negBuilder ) [inline], [inherited]

**0.15.54.3.5 functionCall()** virtual bool smtrat::parser::AbstractTheory::functionCall ( const Identifier & , const std::vector< types::TermType > & , types::TermType & , TheoryError & errors ) [inline], [virtual], [inherited]

Resolve another unknown function call.

Reimplemented in [smtrat::parser::UninterpretedTheory](#), [smtrat::parser::CoreTheory](#), [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

**0.15.54.3.6 handleDistinct()** virtual bool smtrat::parser::AbstractTheory::handleDistinct ( const std::vector< types::TermType > & , types::TermType & , TheoryError & errors ) [inline], [virtual], [inherited]

Resolve a distinct operator.

Reimplemented in [smtrat::parser::CoreTheory](#), [smtrat::parser::BitvectorTheory](#), [smtrat::parser::ArithmeticTheory](#), and [smtrat::parser::UninterpretedTheory](#).

**0.15.54.3.7 handleITE()** virtual bool smtrat::parser::AbstractTheory::handleITE ( const FormulaT & , const types::TermType & , const types::TermType & , types::TermType & , TheoryError & errors ) [inline], [virtual], [inherited]

Resolve an if-then-else operator.

Reimplemented in [smtrat::parser::UninterpretedTheory](#), [smtrat::parser::CoreTheory](#), [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

**0.15.54.3.8 instantiate()** virtual bool smtrat::parser::AbstractTheory::instantiate ( const types::VariableType & , const types::TermType & , types::TermType & , TheoryError & errors ) [inline], [virtual], [inherited]

Instantiate a variable within a term.

Reimplemented in [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

```
0.15.54.3.9 refreshVariable() virtual bool smtrat::parser::AbstractTheory::refreshVariable (
 const types::VariableType & ,
 types::VariableType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Reimplemented in [smtrat::parser::BitvectorTheory](#).

```
0.15.54.3.10 resolveSymbol() virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
 const Identifier & ,
 types::TermType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Resolve a symbol that was not declared within the [ParserState](#).

This might be a symbol that actually uses indices, for example bitvector constants.

Reimplemented in [smtrat::parser::BitvectorTheory](#).

#### 0.15.54.4 Field Documentation

```
0.15.54.4.1 state ParserState* smtrat::parser::AbstractTheory::state [inherited]
```

### 0.15.55 Minisat::BoolOption Class Reference

#include <Options.h>

#### Public Member Functions

- [BoolOption](#) (const char \*c, const char \*n, const char \*d, bool v)
- [operator bool](#) (void) const
- [BoolOption & operator=](#) (bool b)
- virtual void [help](#) (bool verbose=false)

#### Static Protected Member Functions

- static [vec< Option \\* > & getOptionList](#) ()
- static const char \*& [getUsageString](#) ()
- static const char \*& [getHelpPrefixString](#) ()

#### Protected Attributes

- const char \* [name](#)
- const char \* [description](#)
- const char \* [category](#)
- const char \* [type\\_name](#)

#### 0.15.55.1 Constructor & Destructor Documentation

```
0.15.55.1.1 BoolOption() Minisat::BoolOption::BoolOption (
 const char * c,
 const char * n,
 const char * d,
 bool v) [inline]
```

### 0.15.55.2 Member Function Documentation

**0.15.55.2.1 `getHelpPrefixString()`** static const char& Minisat::Option::getHelpPrefixString ( )  
[inline], [static], [protected], [inherited]

**0.15.55.2.2 `getOptionList()`** static vec<Option\*>& Minisat::Option::getOptionList ( ) [inline],  
[static], [protected], [inherited]

**0.15.55.2.3 `getUsageString()`** static const char& Minisat::Option::getUsageString ( ) [inline],  
[static], [protected], [inherited]

**0.15.55.2.4 `help()`** virtual void Minisat::BoolOption::help ( bool verbose = false ) [inline], [virtual]  
Implements [Minisat::Option](#).

**0.15.55.2.5 `operator bool()`** Minisat::BoolOption::operator bool ( void ) const [inline]

**0.15.55.2.6 `operator=()`** BoolOption& Minisat::BoolOption::operator= ( bool b ) [inline]

### 0.15.55.3 Field Documentation

**0.15.55.3.1 `category`** const char\* Minisat::Option::category [protected], [inherited]

**0.15.55.3.2 `description`** const char\* Minisat::Option::description [protected], [inherited]

**0.15.55.3.3 `name`** const char\* Minisat::Option::name [protected], [inherited]

**0.15.55.3.4 `type_name`** const char\* Minisat::Option::type\_name [protected], [inherited]

## 0.15.56 smrat::BoolUEQRewriter Struct Reference

```
#include <BoolUEQRewriter.h>
```

### Public Member Functions

- `BoolUEQRewriter (CollectBoolsInUEQs &&collected)`
- `void setCollected (CollectBoolsInUEQs &&collected)`
- `FormulaT rewrite_bool (const FormulaT &formula)`
- `void addFormulas (std::unordered_multimap<FormulaT, FormulaT> &formulaMap)`
- `void addDomainConstraints (std::unordered_multimap<FormulaT, FormulaT> &formulaMap)`

### 0.15.56.1 Constructor & Destructor Documentation

**0.15.56.1.1 `BoolUEQRewriter()`** `smtrat::BoolUEQRewriter::BoolUEQRewriter ( CollectBoolsInUEQs && collected ) [inline]`

### 0.15.56.2 Member Function Documentation

**0.15.56.2.1 `addDomainConstraints()`** `void smtrat::BoolUEQRewriter::addDomainConstraints ( std::unordered_multimap< FormulaT, FormulaT > & formulaMap ) [inline]`

**0.15.56.2.2 `addFormulas()`** `void smtrat::BoolUEQRewriter::addFormulas ( std::unordered_multimap< FormulaT, FormulaT > & formulaMap ) [inline]`

**0.15.56.2.3 `rewrite_bool()`** `FormulaT smtrat::BoolUEQRewriter::rewrite_bool ( const FormulaT & formula ) [inline]`

**0.15.56.2.4 `setCollected()`** `void smtrat::BoolUEQRewriter::setCollected ( CollectBoolsInUEQs && collected ) [inline]`

## 0.15.57 `smtrat::cadcells::datastructures::Bound` Class Reference

`Bound` type of a [SymbolicInterval](#).

```
#include <roots.h>
```

### Public Member Functions

- `bool is_infy () const`
- `bool is_strict () const`
- `bool is_weak () const`
- `const RootFunction & value () const`
- `void set_weak ()`
- `bool operator== (const Bound &other) const`
- `bool operator!= (const Bound &other) const`

### Static Public Member Functions

- `static Bound infy ()`
- `static Bound strict (RootFunction value)`
- `static Bound weak (RootFunction value)`

### 0.15.57.1 Detailed Description

`Bound` type of a [SymbolicInterval](#).

### 0.15.57.2 Member Function Documentation

**0.15.57.2.1 `infy()`** `static Bound smtrat::cadcells::datastructures::Bound::infy () [inline], [static]`

**0.15.57.2.2 `is_infty()`** `bool smtrat::cadcells::datastructures::Bound::is_infty () const [inline]`

**0.15.57.2.3 `is_strict()`** `bool smtrat::cadcells::datastructures::Bound::is_strict () const [inline]`

**0.15.57.2.4 `is_weak()`** `bool smtrat::cadcells::datastructures::Bound::is_weak () const [inline]`

**0.15.57.2.5 `operator"!="()`** `bool smtrat::cadcells::datastructures::Bound::operator!= (const Bound & other) const [inline]`

**0.15.57.2.6 `operator==( )`** `bool smtrat::cadcells::datastructures::Bound::operator== (const Bound & other) const [inline]`

**0.15.57.2.7 `set_weak()`** `void smtrat::cadcells::datastructures::Bound::set_weak () [inline]`

**0.15.57.2.8 `strict()`** `static Bound smtrat::cadcells::datastructures::Bound::strict (RootFunction value) [inline], [static]`

**0.15.57.2.9 `value()`** `const RootFunction& smtrat::cadcells::datastructures::Bound::value () const [inline]`

**0.15.57.2.10 `weak()`** `static Bound smtrat::cadcells::datastructures::Bound::weak (RootFunction value) [inline], [static]`

## 0.15.58 `smtrat::lra::Bound< T1, T2 >` Class Template Reference

Stores a bound, which could be an upper " $\leq b$ " or a lower bound " $\geq b$ " for a bound value  $b$ .

#include <Bound.h>

### Data Structures

- struct `Info`

*Stores some additional information for a bound.*

### Public Types

- enum `Type` { `LOWER` , `UPPER` , `EQUAL` }  
*The type of the bound: `LOWER` = " $\geq$ ", `UPPER` = " $\leq$ ", `EQUAL` = " $=$ ".*
- typedef `carl::PointerSet< Bound< T1, T2 > > BoundSet`  
*A set of pointer to bounds.*

### Public Member Functions

- `Bound ()=delete`
- `Bound (const Value< T1 > *const _limit, Variable< T1, T2 > *const _var, Type _type, const FormulaT &_constraint, Bound< T1, T2 >::Info *_boundInfo=NULL, bool _deduced=false)`
- `~Bound ()`
- `bool operator> (const Value< T1 > &_value) const`
- `bool operator== (const Value< T1 > &_value) const`

- bool `operator<` (const `Value< T1 >` &`_value`) const
- bool `operator<` (const `Bound &_bound`) const
- bool `operator>` (const `Bound &_bound`) const
- bool `operator==` (const `T1 &_a`) const
- bool `operator>` (const `T1 &_a`) const
- bool `operator<` (const `T1 &_a`) const
- bool `operator>=` (const `T1 &_a`) const
- bool `operator<=` (const `T1 &_a`) const
- const std::string `toString` () const
- void `print` (bool `_withOrigins=false`, std::ostream &`_out=std::cout`, bool `_printTypebool=false`) const
- bool `deduced` () const
- bool `markedAsDeleted` () const
- void `markAsDeleted` () const
- void `unmarkAsDeleted` () const
- const `Value< T1 >` & `limit` () const
- const `Value< T1 >` \* `pLimit` () const
- bool `isInfinite` () const
- `Variable< T1, T2 >` \* `pVariable` () const
- const `Variable< T1, T2 >` & `variable` () const
- `Type type` () const
- bool `isWeak` () const
- bool `isUpperBound` () const
- bool `isLowerBound` () const
- const `FormulaT & asConstraint` () const
- bool `hasNeqRepresentation` () const
- const `FormulaT & neqRepresentation` () const
- void `setNeqRepresentation` (const `FormulaT &_constraint`) const
- void `resetNeqRepresentation` () const
- void `boundExists` () const
- void `setComplement` (const `Bound< T1, T2 >` \*`_complement`) const
- bool `exists` () const
- const std::shared\_ptr< std::vector< `FormulaT` > > & `pOrigins` () const
- const std::vector< `FormulaT` > & `origins` () const
- bool `isActive` () const
- bool `isComplementActive` () const
- bool `isUnassigned` () const
- `Info * pInfo` () const
- bool `isSatisfied` (const `T1 &_delta`) const
- bool `operator>=` (const `Value< T1 >` &`v`) const
- bool `operator<=` (const `Value< T1 >` &`v`) const

## Friends

- template<typename T3 , typename T4 >  
std::ostream & `operator<<` (std::ostream &`_ostream`, const `Bound< T3, T4 >` &`_bound`)

### 0.15.58.1 Detailed Description

```
template<typename T1, typename T2>
class smrat::ira::Bound< T1, T2 >
```

Stores a bound, which could be an upper " $\leq b$ " or a lower bound " $\geq b$ " for a bound value  $b$ .  
 In the case that " $\leq b$ " and " $\geq b$ " hold this bound is a so called equal bound represented by " $\equiv b$ ".

### 0.15.58.2 Member Typedef Documentation

**0.15.58.2.1 BoundSet** template<typename T1 , typename T2 >  
typedef carl::PointerSet<Bound<T1, T2> > smtrat::lra::Bound< T1, T2 >::BoundSet  
A set of pointer to bounds.

### 0.15.58.3 Member Enumeration Documentation

**0.15.58.3.1 Type** template<typename T1 , typename T2 >  
enum smtrat::lra::Bound::Type  
The type of the bound: LOWER = ">=", UPPER = "<=", EQUAL "==".

#### Enumerator

|       |  |
|-------|--|
| LOWER |  |
| UPPER |  |
| EQUAL |  |

### 0.15.58.4 Constructor & Destructor Documentation

**0.15.58.4.1 Bound() [1/2]** template<typename T1 , typename T2 >  
smtrat::lra::Bound< T1, T2 >::Bound ( ) [delete]

**0.15.58.4.2 Bound() [2/2]** template<typename T1 , typename T2 >  
smtrat::lra::Bound< T1, T2 >::Bound (   
    const Value< T1 > \*const \_limit,  
    Variable< T1, T2 > \*const \_var,  
    Type \_type,  
    const FormulaT & \_constraint,  
    Bound< T1, T2 >::Info \* \_boundInfo = NULL,  
    bool \_deduced = false )

#### Parameters

|             |  |
|-------------|--|
| _limit      |  |
| _var        |  |
| _type       |  |
| _constraint |  |
| _boundInfo  |  |
| _deduced    |  |

**0.15.58.4.3 ~Bound()** template<typename T1 , typename T2 >  
smtrat::lra::Bound< T1, T2 >::~Bound ( )

### 0.15.58.5 Member Function Documentation

**0.15.58.5.1 asConstraint()** template<typename T1 , typename T2 >  
const FormulaT& smtrat::lra::Bound< T1, T2 >::asConstraint ( ) const [inline]

Returns

**0.15.58.5.2 boundExists()** template<typename T1 , typename T2 >  
void **smtrat::lra::Bound**< T1, T2 >::boundExists ( ) const [inline]

**0.15.58.5.3 deduced()** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound**< T1, T2 >::deduced ( ) const [inline]

Returns

**0.15.58.5.4 exists()** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound**< T1, T2 >::exists ( ) const [inline]

**0.15.58.5.5 hasNeqRepresentation()** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound**< T1, T2 >::hasNeqRepresentation ( ) const [inline]

Returns

**0.15.58.5.6 isActive()** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound**< T1, T2 >::isActive ( ) const [inline]

Returns

**0.15.58.5.7 isComplementActive()** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound**< T1, T2 >::isComplementActive ( ) const [inline]

Returns

**0.15.58.5.8 isInfinite()** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound**< T1, T2 >::isInfinite ( ) const [inline]

Returns

**0.15.58.5.9 isLowerBound()** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound**< T1, T2 >::isLowerBound ( ) const [inline]

Returns

**0.15.58.5.10 `isSatisfied()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::isSatisfied ( const T1 & \_delta ) const [inline]

**0.15.58.5.11 `isUnassigned()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::isUnassigned ( ) const [inline]

**0.15.58.5.12 `isUpperBound()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::isUpperBound ( ) const [inline]

**Returns**

**0.15.58.5.13 `isWeak()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::isWeak ( ) const [inline]

**Returns**

**0.15.58.5.14 `limit()`** template<typename T1 , typename T2 >  
const Value<T1>& smtrat::lra::Bound< T1, T2 >::limit ( ) const [inline]

**Returns**

**0.15.58.5.15 `markAsDeleted()`** template<typename T1 , typename T2 >  
void smtrat::lra::Bound< T1, T2 >::markAsDeleted ( ) const [inline]

**Returns**

**0.15.58.5.16 `markedAsDeleted()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::markedAsDeleted ( ) const [inline]

**Returns**

**0.15.58.5.17 `neqRepresentation()`** template<typename T1 , typename T2 >  
const FormulaT& smtrat::lra::Bound< T1, T2 >::neqRepresentation ( ) const [inline]

**Returns**

**0.15.58.5.18 `operator<()` [1/3]** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::operator< ( const Bound< T1, T2 > & \_bound ) const

**Parameters**

|               |  |
|---------------|--|
| <i>_bound</i> |  |
|---------------|--|

**Returns**

**0.15.58.5.19 operator<() [2/3]** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound<** T1, T2 **>::operator<** (  
    const T1 & *\_a* ) const

**Parameters**

|                      |  |
|----------------------|--|
| $\leftrightarrow$    |  |
| $\_ \leftrightarrow$ |  |
| <i>a</i>             |  |

**Returns**

**0.15.58.5.20 operator<() [3/3]** template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound<** T1, T2 **>::operator<** (  
    const Value< T1 > & *\_value* ) const

**Parameters**

|               |  |
|---------------|--|
| <i>_value</i> |  |
|---------------|--|

**Returns**

**0.15.58.5.21 operator<=()** [1/2] template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound<** T1, T2 **>::operator<=** (  
    const T1 & *\_a* ) const

**Parameters**

|                      |  |
|----------------------|--|
| $\leftrightarrow$    |  |
| $\_ \leftrightarrow$ |  |
| <i>a</i>             |  |

**Returns**

**0.15.58.5.22 operator<=()** [2/2] template<typename T1 , typename T2 >  
bool **smtrat::lra::Bound<** T1, T2 **>::operator<=** (  
    const Value< T1 > & *v* ) const [inline]

Returns

**0.15.58.5.23 operator==( ) [1/2]** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::operator== (  
    const T1 & \_a ) const

Parameters

|                      |  |
|----------------------|--|
| $\leftrightarrow$    |  |
| $\_ \leftrightarrow$ |  |
| <i>a</i>             |  |

Returns

**0.15.58.5.24 operator==( ) [2/2]** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::operator== (  
    const Value< T1 > & \_value ) const

Parameters

|               |  |
|---------------|--|
| <i>_value</i> |  |
|---------------|--|

Returns

**0.15.58.5.25 operator>() [1/3]** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::operator> (  
    const Bound< T1, T2 > & \_bound ) const

Parameters

|               |  |
|---------------|--|
| <i>_bound</i> |  |
|---------------|--|

Returns

**0.15.58.5.26 operator>() [2/3]** template<typename T1 , typename T2 >  
bool smtrat::lra::Bound< T1, T2 >::operator> (  
    const T1 & \_a ) const

Parameters

|                      |  |
|----------------------|--|
| $\leftrightarrow$    |  |
| $\_ \leftrightarrow$ |  |
| <i>a</i>             |  |

Returns

```
0.15.58.5.27 operator>() [3/3] template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator> (
 const Value< T1 > & _value) const
```

Parameters

|                     |  |
|---------------------|--|
| <code>_value</code> |  |
|---------------------|--|

Returns

```
0.15.58.5.28 operator>=() [1/2] template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator>= (
 const T1 & _a) const
```

Parameters

|                      |  |
|----------------------|--|
| $\leftrightarrow$    |  |
| $\_ \leftrightarrow$ |  |
| <code>a</code>       |  |

Returns

```
0.15.58.5.29 operator>=() [2/2] template<typename T1 , typename T2 >
bool smtrat::lra::Bound< T1, T2 >::operator>= (
 const Value< T1 > & v) const [inline]
```

Returns

```
0.15.58.5.30 origins() template<typename T1 , typename T2 >
const std::vector<FormulaT>& smtrat::lra::Bound< T1, T2 >::origins () const [inline]
```

Returns

```
0.15.58.5.31 pInfo() template<typename T1 , typename T2 >
Info* smtrat::lra::Bound< T1, T2 >::pInfo () const [inline]
```

Returns

**0.15.58.5.32 pLimit()** template<typename T1 , typename T2 >  
const `Value<T1>*` `smtrat::lra::Bound< T1, T2 >::pLimit` ( ) const [inline]

Returns

**0.15.58.5.33 pOrigins()** template<typename T1 , typename T2 >  
const `std::shared_ptr<std::vector<FormulaT> >&` `smtrat::lra::Bound< T1, T2 >::pOrigins` ( )  
const [inline]

Returns

**0.15.58.5.34 print()** template<typename T1 , typename T2 >  
void `smtrat::lra::Bound< T1, T2 >::print` (  
    bool `_withOrigins` = `false`,  
    `std::ostream & _out` = `std::cout`,  
    bool `_printTypebool` = `false` ) const

Parameters

|                             |  |
|-----------------------------|--|
| <code>_withOrigins</code>   |  |
| <code>_out</code>           |  |
| <code>_printTypebool</code> |  |

**0.15.58.5.35 pVariable()** template<typename T1 , typename T2 >  
`Variable<T1, T2>*` `smtrat::lra::Bound< T1, T2 >::pVariable` ( ) const [inline]

Returns

**0.15.58.5.36 resetNeqRepresentation()** template<typename T1 , typename T2 >  
void `smtrat::lra::Bound< T1, T2 >::resetNeqRepresentation` ( ) const [inline]

**0.15.58.5.37 setComplement()** template<typename T1 , typename T2 >  
void `smtrat::lra::Bound< T1, T2 >::setComplement` (  
    const `Bound< T1, T2 > *` `_complement` ) const [inline]

**0.15.58.5.38 setNeqRepresentation()** template<typename T1 , typename T2 >  
void `smtrat::lra::Bound< T1, T2 >::setNeqRepresentation` (  
    const `FormulaT &` `_constraint` ) const [inline]

Parameters

|                          |  |
|--------------------------|--|
| <code>_constraint</code> |  |
|--------------------------|--|

```
0.15.58.5.39 toString() template<typename T1 , typename T2 >
const std::string smtrat::lra::Bound< T1, T2 >::toString () const
```

Returns

```
0.15.58.5.40 type() template<typename T1 , typename T2 >
Type smtrat::lra::Bound< T1, T2 >::type () const [inline]
```

Returns

```
0.15.58.5.41 unmarkAsDeleted() template<typename T1 , typename T2 >
void smtrat::lra::Bound< T1, T2 >::unmarkAsDeleted () const [inline]
```

Returns

```
0.15.58.5.42 variable() template<typename T1 , typename T2 >
const Variable<T1, T2>& smtrat::lra::Bound< T1, T2 >::variable () const [inline]
```

Returns

## 0.15.58.6 Friends And Related Function Documentation

```
0.15.58.6.1 operator<< template<typename T1 , typename T2 >
template<typename T3 , typename T4 >
std::ostream& operator<< (
 std::ostream & _ostream,
 const Bound< T3, T4 > & _bound) [friend]
```

Parameters

|                             |  |
|-----------------------------|--|
| <code>_withOrigins</code>   |  |
| <code>_out</code>           |  |
| <code>_printTypebool</code> |  |

## 0.15.59 smtrat::mcsat::fm::Bound Struct Reference

```
#include <ConflictGenerator.h>
```

### Public Member Functions

- `Bound (ConstraintT constr, Poly p, Poly q, Rational r, bool neg)`

**Data Fields**

- `ConstraintT constr`
- `Poly p`
- `Poly q`
- `Rational r`
- bool `neg`

**Friends**

- `std::ostream & operator<< (std::ostream &os, const Bound &dt)`

**0.15.59.1 Constructor & Destructor Documentation**

**0.15.59.1.1 `Bound()`** smtrat::mcsat::fm::Bound::Bound (

```
 ConstraintT constr,
 Poly p,
 Poly q,
 Rational r,
 bool neg) [inline]
```

**0.15.59.2 Friends And Related Function Documentation**

**0.15.59.2.1 `operator<<`** std::ostream& operator<< (

```
 std::ostream & os,
 const Bound & dt) [friend]
```

**0.15.59.3 Field Documentation**

**0.15.59.3.1 `constr`** ConstraintT smtrat::mcsat::fm::Bound::constr

**0.15.59.3.2 `neg`** bool smtrat::mcsat::fm::Bound::neg

**0.15.59.3.3 `p`** Poly smtrat::mcsat::fm::Bound::p

**0.15.59.3.4 `q`** Poly smtrat::mcsat::fm::Bound::q

**0.15.59.3.5 `r`** Rational smtrat::mcsat::fm::Bound::r

**0.15.60 smtrat::vb::Bound< T > Class Template Reference**

Class for the bound of a variable.

```
#include <VariableBounds.h>
```

## Public Types

- enum `Type` {  
    `STRICT_LOWER_BOUND` = 0, `WEAK_LOWER_BOUND` = 1, `EQUAL_BOUND` = 2, `WEAK_UPPER_BOUND` = 3,  
    `STRICT_UPPER_BOUND` = 4}

*The type of a bounds.*

## Public Member Functions

- `Bound (Rational *const _limit, Variable< T > *const _variable, Type _type)`  
*Constructs this bound.*
- `~Bound ()`  
*Destructs this bound.*
- `bool operator< (const Bound< T > &_bound) const`  
*Checks whether the this bound is smaller than the given one.*
- `void print (std::ostream &_out, bool _withRelation=false) const`  
*Prints this bound on the given stream.*
- `const Rational & limit () const`
- `const Rational * pLimit () const`
- `bool isInfinite () const`
- `Type type () const`
- `bool isUpperBound () const`
- `bool isLowerBound () const`
- `Variable< T > * pVariable () const`
- `const Variable< T > & variable () const`
- `bool isActive () const`
- `bool activate (const T &_origin) const`  
*Adds an origin to this bound.*
- `bool deactivate (const T &_origin) const`  
*Removes an origin from this bound.*
- `const std::set< T, std::less< T, false > > & origins () const`

## Friends

- `template<typename T1>`  
`std::ostream & operator<< (std::ostream &_out, const Bound< T1 > &_bound)`  
*Prints the bound on the given output stream.*

### 0.15.60.1 Detailed Description

```
template<typename T>
class smrat::vb::Bound< T >
```

Class for the bound of a variable.

### 0.15.60.2 Member Enumeration Documentation

**Enumerator**

**0.15.60.2.1 Type** template<typename T >  
enum **smtrat::vb::Bound::Type**  
The type of a bounds.

**Enumerator**

|                    |
|--------------------|
| STRICT_LOWER_BOUND |
| WEAK_LOWER_BOUND   |
| EQUAL_BOUND        |
| WEAK_UPPER_BOUND   |
| STRICT_UPPER_BOUND |

**0.15.60.3 Constructor & Destructor Documentation**

**0.15.60.3.1 Bound()** template<typename T >  
**smtrat::vb::Bound< T >::Bound** (  
    **Rational** \*const *\_limit*,  
    **Variable**< T > \*const *\_variable*,  
    **Type** *\_type* )

Constructs this bound.

**Parameters**

|                  |                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------|
| <i>_limit</i>    | A pointer to the limit (rational) of the bound. It is NULL, if the limit is not finite.              |
| <i>_variable</i> | The variable to which this bound belongs.                                                            |
| <i>_type</i>     | The type of the bound (either it is an equal bound or it is weak resp. strict and upper resp. lower) |

**0.15.60.3.2 ~Bound()** template<typename T >  
**smtrat::vb::Bound< T >::~Bound** ( )

Destructs this bound.

**0.15.60.4 Member Function Documentation**

**0.15.60.4.1 activate()** template<typename T >  
bool **smtrat::vb::Bound< T >::activate** (  
    const T & *\_origin* ) const [inline]

Adds an origin to this bound.

**Parameters**

|                |                    |
|----------------|--------------------|
| <i>_origin</i> | The origin to add. |
|----------------|--------------------|

**Returns**

true, if this has activated this bound; false, if the bound was already active before.

```
0.15.60.4.2 deactivate() template<typename T >
bool smtrat::vb::Bound< T >::deactivate (
 const T & _origin) const [inline]
Removes an origin from this bound.
```

**Parameters**

|                |                    |
|----------------|--------------------|
| <i>_origin</i> | The origin to add. |
|----------------|--------------------|

**Returns**

true, if this has deactivated this bound; false, if the bound was already inactive before.

```
0.15.60.4.3 isActive() template<typename T >
bool smtrat::vb::Bound< T >::isActive () const [inline]
```

**Returns**

true, if this bound is active, which means that there is at least one origin for it left. Note, that origins can be removed belatedly. false, otherwise.

```
0.15.60.4.4 isInfinite() template<typename T >
bool smtrat::vb::Bound< T >::isInfinite () const [inline]
```

**Returns**

true, if the bound infinite; false, otherwise.

```
0.15.60.4.5 isLowerBound() template<typename T >
bool smtrat::vb::Bound< T >::isLowerBound () const [inline]
```

**Returns**

true, if the bound is a lower bound; false, otherwise.

```
0.15.60.4.6 isUpperBound() template<typename T >
bool smtrat::vb::Bound< T >::isUpperBound () const [inline]
```

**Returns**

true, if the bound is an upper bound; false, otherwise.

```
0.15.60.4.7 limit() template<typename T >
const Rational& smtrat::vb::Bound< T >::limit () const [inline]
```

**Returns**

A constant reference to the value of the limit. Note, that it must be ensured that the bound is finite before calling this method.

```
0.15.60.4.8 operator<() template<typename T >
bool smtrat::vb::Bound< T >::operator< (
 const Bound< T > & _bound) const
```

Checks whether the this bound is smaller than the given one.

**Parameters**

|               |                            |
|---------------|----------------------------|
| <i>_bound</i> | The bound to compare with. |
|---------------|----------------------------|

**Returns**

true, if for this bound (A) and the given bound (B) it holds that: A is not inf and B is inf, A smaller than B, where A and B are rationals, A equals B, where A and B are rationals, but A is an equal bound but B is not; false, otherwise.

**0.15.60.4.9 `origins()`** `template<typename T >`

```
const std::set<T, carl::less<T, false> >& smtrat::vb::Bound< T >::origins () const [inline]
```

**Returns**

A constant reference to the set of origins of this bound.

**0.15.60.4.10 `pLimit()`** `template<typename T >`

```
const Rational* smtrat::vb::Bound< T >::pLimit () const [inline]
```

**Returns**

A pointer to the limit of this bound.

**0.15.60.4.11 `print()`** `template<typename T >`

```
void smtrat::vb::Bound< T >::print (
 std::ostream & _out,
 bool _withRelation = false) const
```

Prints this bound on the given stream.

**Parameters**

|                      |                                                                                        |
|----------------------|----------------------------------------------------------------------------------------|
| <i>_out</i>          | The stream to print on.                                                                |
| <i>_withRelation</i> | A flag indicating whether to print also a relation symbol in front of the bound value. |

**0.15.60.4.12 `pVariable()`** `template<typename T >`

```
Variable<T>* smtrat::vb::Bound< T >::pVariable () const [inline]
```

**Returns**

A pointer to the variable wrapper considered by this bound.

**0.15.60.4.13 `type()`** `template<typename T >`

```
Type smtrat::vb::Bound< T >::type () const [inline]
```

**Returns**

The type of this bound.

```
0.15.60.4.14 variable() template<typename T >
const Variable<T>& smrat::vb::Bound< T >::variable () const [inline]
```

**Returns**

A constant reference to the variable wrapper considered by this bound.

**0.15.60.5 Friends And Related Function Documentation**

```
0.15.60.5.1 operator<< template<typename T >
template<typename T1 >
std::ostream& operator<< (
 std::ostream & _out,
 const Bound< T1 > & _bound) [friend]
```

Prints the bound on the given output stream.

**Parameters**

|                     |                                |
|---------------------|--------------------------------|
| <code>_out</code>   | The output stream to print on. |
| <code>_bound</code> | The bound to print.            |

**Returns**

The output stream after printing the bound on it.

**0.15.61 smrat::Branching Struct Reference**

Stores all necessary information of an branch, which can be used to detect probable infinite loop of branchings.

```
#include <Module.h>
```

**Public Member Functions**

- **Branching** (const typename Poly::PolyType &`_polynomial`, const Rational &`_value`)  
*Constructor.*

**Data Fields**

- Poly::PolyType **mPolynomial**  
*The polynomial to branch at.*
- Rational **mValue**  
*The value to branch the polynomial at.*
- size\_t **mRepetitions**  
*The number of repetitions of the branching.*
- int **mIncreasing**  
*Stores whether this the branching of the polynomial has been on an increasing sequence of values ( $>0$ ), or on a decreasing sequence of values ( $<0$ ).*

**0.15.61.1 Detailed Description**

Stores all necessary information of an branch, which can be used to detect probable infinite loop of branchings.

**0.15.61.2 Constructor & Destructor Documentation**

```
0.15.61.2.1 Branching() smtrat::Branching::Branching (
 const typename Poly::PolyType & _polynomial,
 const Rational & _value) [inline]
```

Constructor.

#### Parameters

|                          |                                        |
|--------------------------|----------------------------------------|
| <code>_polynomial</code> | The polynomial to branch at.           |
| <code>_value</code>      | The value to branch the polynomial at. |

### 0.15.61.3 Field Documentation

#### 0.15.61.3.1 mIncreasing int smtrat::Branching::mIncreasing

Stores whether this the branching of the polynomial has been on an increasing sequence of values ( $>0$ ), or on a decreasing sequence of values ( $<0$ ).

It is initially zero.

#### 0.15.61.3.2 mPolynomial Poly::PolyType smtrat::Branching::mPolynomial

The polynomial to branch at.

#### 0.15.61.3.3 mRepetitions size\_t smtrat::Branching::mRepetitions

The number of repetitions of the branching.

#### 0.15.61.3.4 mValue Rational smtrat::Branching::mValue

The value to branch the polynomial at.

## 0.15.62 smtrat::BVAnnotation Class Reference

```
#include <IntBlastModule.h>
```

### Public Member Functions

- `BVAnnotation ()`
- `BVAnnotation (std::size_t _width, bool _signed, Integer _offset=0)`
- `std::size_t width () const`
- `bool isSigned () const`
- `bool is_constant () const`
- `bool hasOffset () const`
- `BVAnnotation withOffset (Integer _newOffset) const`
- `BVAnnotation withWidth (std::size_t _width) const`
- `const Integer & offset () const`
- `const carl::Interval< Integer > & bounds () const`
- `const Integer & lower_bound () const`
- `const Integer & upper_bound () const`

### Static Public Member Functions

- static `BVAnnotation forSum (BVAnnotation _summand1, BVAnnotation _summand2)`
- static `BVAnnotation forProduct (BVAnnotation _factor1, BVAnnotation _factor2)`

**Friends**

- std::ostream & `operator<<` (std::ostream &\_out, const `BVAnnotation` &\_type)

**0.15.62.1 Constructor & Destructor Documentation****0.15.62.1.1 `BVAnnotation()`** [1/2] smtrat::BVAnnotation::BVAnnotation ( ) [inline]**0.15.62.1.2 `BVAnnotation()`** [2/2] smtrat::BVAnnotation::BVAnnotation ( std::size\_t \_width, bool \_signed, Integer \_offset = 0 ) [inline]**0.15.62.2 Member Function Documentation****0.15.62.2.1 `bounds()`** const carl::Interval<Integer>& smtrat::BVAnnotation::bounds ( ) const [inline]**0.15.62.2.2 `forProduct()`** static `BVAnnotation` smtrat::BVAnnotation::forProduct ( `BVAnnotation` \_factor1, `BVAnnotation` \_factor2 ) [inline], [static]**0.15.62.2.3 `forSum()`** static `BVAnnotation` smtrat::BVAnnotation::forSum ( `BVAnnotation` \_summand1, `BVAnnotation` \_summand2 ) [inline], [static]**0.15.62.2.4 `hasOffset()`** bool smtrat::BVAnnotation::hasOffset ( ) const [inline]**0.15.62.2.5 `is_constant()`** bool smtrat::BVAnnotation::is\_constant ( ) const [inline]**0.15.62.2.6 `isSigned()`** bool smtrat::BVAnnotation::isSigned ( ) const [inline]**0.15.62.2.7 `lower_bound()`** const Integer& smtrat::BVAnnotation::lower\_bound ( ) const [inline]**0.15.62.2.8 `offset()`** const Integer& smtrat::BVAnnotation::offset ( ) const [inline]**0.15.62.2.9 `upper_bound()`** const Integer& smtrat::BVAnnotation::upper\_bound ( ) const [inline]**0.15.62.2.10 `width()`** std::size\_t smtrat::BVAnnotation::width ( ) const [inline]

**0.15.62.2.11 withOffset()** `BVAnnotation smtrat::BVAnnotation::withOffset (`  
`Integer _newOffset ) const [inline]`

**0.15.62.2.12 withWidth()** `BVAnnotation smtrat::BVAnnotation::withWidth (`  
`std::size_t _width ) const [inline]`

### 0.15.62.3 Friends And Related Function Documentation

**0.15.62.3.1 operator<<** `std::ostream& operator<< (`  
`std::ostream & _out,`  
`const BVAnnotation & _type ) [friend]`

## 0.15.63 smtrat::BVDirectEncoder Class Reference

```
#include <BVDirectEncoder.h>
```

### Public Member Functions

- const `FormulaSetT & encode (const FormulaT &_inputFormula)`
- const `std::set< carl::Variable > & introducedVariables () const`
- const `carl::FastMap< BitVec, Variables > bitvectorBlastings () const`
- `BVDirectEncoder ()`
- `~BVDirectEncoder ()`

### 0.15.63.1 Constructor & Destructor Documentation

**0.15.63.1.1 BVDirectEncoder()** `smtrat::BVDirectEncoder::BVDirectEncoder ( ) [inline]`

**0.15.63.1.2 ~BVDirectEncoder()** `smtrat::BVDirectEncoder::~BVDirectEncoder ( ) [inline]`

### 0.15.63.2 Member Function Documentation

**0.15.63.2.1 bitvectorBlastings()** `const carl::FastMap<BitVec, Variables> smtrat::BVDirectEncoder::bitvectorBlastings ( ) const [inline]`

**0.15.63.2.2 encode()** `const FormulaSetT& smtrat::BVDirectEncoder::encode (`  
`const FormulaT & _inputFormula ) [inline]`

**0.15.63.2.3 introducedVariables()** `const std::set<carl::Variable>& smtrat::BVDirectEncoder::introducedVariables ( ) const [inline]`

## 0.15.64 smtrat::BVModule< Settings > Class Template Reference

```
#include <BVModule.h>
```

## Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

## Public Member Functions

- `std::string moduleName () const`
- `BVModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~BVModule ()`
- `bool informCore (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `bool addCore (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `void removeCore (ModuleInput::const_iterator _subformula)`  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- `void updateModel () const`  
*Updates the current assignment into the model.*
- `Answer checkCore ()`  
*Checks the received formula for consistency.*
- `bool inform (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void deinform (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- `bool add (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
*Checks the received formula for consistency.*
- `virtual void remove (ModuleInput::const_iterator _subformula)`  
*Removes everything related to the given sub-formula of the received formula.*
- `virtual void updateAllModels ()`  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`  
*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`  
*Sets the priority of this module to get a thread for running its check procedure.*
- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`
- `const std::vector< Model > & allModels () const`

- const std::vector< FormulaSetT > & **infeasibleSubsets** () const
- const std::vector< Module \* > & **usedBackends** () const
- const carl::FastSet< FormulaT > & **constraintsToInform** () const
- const carl::FastSet< FormulaT > & **informedConstraints** () const
- void **addLemma** (const FormulaT &\_lemma, const LemmaType &\_lt=LemmaType::NORMAL, const FormulaT &\_preferredFormula=FormulaT(carl::FormulaType::TRUE))
 

*Stores a lemma being a valid formula.*
- bool **hasLemmas** ()
 

*Checks whether this module or any of its backends provides any lemmas.*
- void **clearLemmas** ()
 

*Deletes all yet found lemmas.*
- const std::vector< Lemma > & **lemmas** () const
- ModuleInput::const\_iterator **firstUncheckedReceivedSubformula** () const
- ModuleInput::const\_iterator **firstSubformulaToPass** () const
- void **receivedFormulaChecked** ()
 

*Notifies that the received formulas has been checked.*
- const smtrat::Conditionals & **answerFound** () const
- bool **isPreprocessor** () const
- carl::Variable **objective** () const
- bool **is\_minimizing** () const
- void **excludeNotReceivedVariablesFromModel** () const
 

*Excludes all variables from the current model, which do not occur in the received formula.*
- void **updateLemmas** ()
 

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void **collectTheoryPropagations** ()
- void **collectOrigins** (const FormulaT &\_formula, FormulasT &\_origins) const
 

*Collects the formulas in the given formula, which are part of the received formula.*
- void **collectOrigins** (const FormulaT &\_formula, FormulaSetT &\_origins) const
- bool **isValidInfeasibleSubset** () const
- void **checkInfeasibleSubsetForMinimality** (std::vector< FormulaSetT >::const\_iterator \_infssubset, const std::string &filename="smaller\_muses", unsigned \_maxSizeDifference=1) const
 

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, FormulaT > **getReceivedFormulaSimplified** ()
- void **print** (const std::string &\_initiation="\*\*\*") const
 

*Prints everything relevant of the solver.*
- void **printReceivedFormula** (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of the received formula.*
- void **printPassedFormula** (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of passed formula.*
- void **printInfeasibleSubsets** (const std::string &\_initiation="\*\*\*") const
 

*Prints the infeasible subsets.*
- void **printModel** (std::ostream &\_out=std::cout) const
 

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void **printAllModels** (std::ostream &\_out=std::cout)
 

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void **freeSplittingVariable** (const FormulaT &\_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- size\_t `evaluateBVFormula` (const FormulaT &formula)
- void `transferBackendModel` () const
- virtual void `deinformCore` (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- ModuleInput::iterator `passedFormulaBegin` ()
- ModuleInput::iterator `passedFormulaEnd` ()
- void `addOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const FormulaT & `getOrigins` (ModuleInput::const\_iterator \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const FormulaT &\_formula, FormulasT &\_origins) const
- void `getOrigins` (const FormulaT &\_formula, FormulaSetT &\_origins) const
- std::pair< ModuleInput::iterator, bool > `removeOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)
- std::pair< ModuleInput::iterator, bool > `removeOrigins` (ModuleInput::iterator \_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)
- void `informBackends` (const FormulaT &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const FormulaT &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< ModuleInput::iterator, bool > `addReceivedSubformulaToPassedFormula` (ModuleInput::const\_iterator \_subformula)  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const FormulaT &\_origin) const
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula)  
*Adds the given formula to the passed formula with no origin.*
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)  
*Adds the given formula to the passed formula.*
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula, const FormulaT &\_origin)  
*Adds the given formula to the passed formula.*

- Adds the given formula to the passed formula.
- void `generateTrivialInfeasibleSubset ()`

Stores the trivial infeasible subset being the set of received formulas.
- void `receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

Stores an infeasible subset consisting only of the given received formula.
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin (const std::vector< FormulaT > &_origins) const`
- void `getInfeasibleSubsets ()`

Copies the infeasible subsets of the passed formula.
- std::vector< `FormulaSetT` > `getInfeasibleSubsets (const Module &_backend) const`

Get the infeasible subsets the given backend provides.
- const `Model & backendsModel () const`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void `getBackendsModel () const`
- void `getBackendsAllModels () const`

Stores all models of a backend in the list of all models of this module.
- virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

Runs the backend solvers on the passed formula.
- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void `clearPassedFormula ()`
- std::vector< `FormulaT` > `merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`

Merges the two vectors of sets into the first one.
- size\_t `determine_smallest_origin (const std::vector< FormulaT > &origins) const`
- bool `probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())`
- void `splitUnequalConstraint (const FormulaT &)`

Adds the following lemmas for the given constraint  $p \neq 0$ .
- unsigned `checkModel () const`
- void `addInformationRelevantFormula (const FormulaT &formula)`

Adds a formula to the InformationRelevantFormula.
- const std::set< `FormulaT` > & `getInformationRelevantFormulas ()`

Gets all InformationRelevantFormulas.
- bool `isLemmaLevel (LemmaLevel level)`

Checks if current lemma level is greater or equal to given level.

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector<FormulaSetT> mInfeasibleSubsets`

*Stores the infeasible subsets.*

- `Manager *const mpManager`

*A reference to the manager.*

- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*

- `std::vector<Model> mAllModels`

*Stores all satisfying assignments.*

- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*

- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*

- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*

- `std::vector<TheoryPropagation> mTheoryPropagations`

- `std::atomic<Answer> mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- `Conditionals mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- `std::vector<Module*> mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- `std::vector<Module*> mAllBackends`

*The backends of this module which have been used.*

- `std::vector<Lemma> mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*

- `ModuleInput::iterator mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- `carl::FastSet<FormulaT> mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet<FormulaT> mInformedConstraints`

*Stores the position of the first constraint of which no backend has been informed about.*

- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- `unsigned mSmallerMusesCheckCounter`

*Counter used for the generation of the smt2 files to check for smaller muses.*

- `std::vector<std::size_t> mVariableCounters`

*Maps variables to the number of their occurrences.*

### 0.15.64.1 Member Typedef Documentation

```
0.15.64.1.1 SettingsType template<typename Settings >
typedef Settings smtrat::BVModule< Settings >::SettingsType
```

### 0.15.64.2 Member Enumeration Documentation

**0.15.64.2.1 LemmaType** enum [smtrat::Module::LemmaType](#) : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.64.3 Constructor & Destructor Documentation

```
0.15.64.3.1 BVModule() template<typename Settings >
smtrat::BVModule< Settings >::BVModule (
 const ModuleInput * _formula,
 Conditionals & _conditionals,
 Manager * _manager = NULL)
```

```
0.15.64.3.2 ~BVModule() template<typename Settings >
smtrat::BVModule< Settings >::~BVModule ()
```

### 0.15.64.4 Member Function Documentation

**0.15.64.4.1 add()** bool [smtrat::Module::add](#) (  
 [ModuleInput::const\\_iterator](#) *\_subformula* ) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub-formula to take additionally into account. |
|--------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

```
0.15.64.4.2 addConstraintToInform() void smtrat::Module::addConstraintToInform (
 const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

|                    |                        |
|--------------------|------------------------|
| <i>_constraint</i> | The constraint to add. |
|--------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

```
0.15.64.4.3 addCore() template<typename Settings >
bool smtrat::BVModule< Settings >::addCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.64.4.4 addInformationRelevantFormula() void smtrat::Module::addInformationRelevantFormula (
 const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

```
0.15.64.4.5 addLemma() void smtrat::Module::addLemma (
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

```
0.15.64.4.6 addOrigin() void smtrat::Module::addOrigin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.64.4.7 addReceivedSubformulaToPassedFormula()** std::pair<ModuleInput::iterator,bool>  
smtrat::Module::addReceivedSubformulaToPassedFormula (

    ModuleInput::const\_iterator \_subformula ) [inline], [protected], [inherited]

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

**0.15.64.4.8 addSubformulaToPassedFormula() [1/3]** std::pair<ModuleInput::iterator,bool> smtrat<->  
::Module::addSubformulaToPassedFormula (

    const FormulaT & \_formula ) [inline], [protected], [inherited]

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.64.4.9 addSubformulaToPassedFormula() [2/3]** std::pair<ModuleInput::iterator,bool> smtrat<->  
::Module::addSubformulaToPassedFormula (

    const FormulaT & \_formula,

    const FormulaT & \_origin ) [inline], [protected], [inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

|         |                                                                                                                           |
|---------|---------------------------------------------------------------------------------------------------------------------------|
| _origin | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |
|---------|---------------------------------------------------------------------------------------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.64.4.10 addSubformulaToPassedFormula() [3/3]** std::pair<ModuleInput::iterator,bool> smtrat<->  
::Module::addSubformulaToPassedFormula (

    const FormulaT & \_formula,

    const std::shared\_ptr< std::vector< FormulaT >> & \_origins ) [inline], [protected],

[inherited]

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.64.4.11 `allModels()`** `const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]`

**Returns**

All satisfying assignments, if existent.

**0.15.64.4.12 `anAnswerFound()`** `bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.64.4.13 `answerFound()`** `const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]`

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.64.4.14 `backendsModel()`** `const Model & smtrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.64.4.15 `branchAt()` [1/4]** `bool smtrat::Module::branchAt ( carl::Variable::Arg _var, const Rational & _value, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & _premise = std::vector< FormulaT > () ) [inline], [protected], [inherited]`

```
0.15.64.4.16 branchAt() [2/4] bool smrat::Module::branchAt (
 carl::Variable::Arg _var,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

```
0.15.64.4.17 branchAt() [3/4] bool smrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

```
0.15.64.4.18 branchAt() [4/4] bool smrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.64.4.19 check() Answer smrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.64.4.20 `checkCore()`** `template<typename Settings >`  
`Answer smtrat::BVModule< Settings >::checkCore ( ) [virtual]`  
 Checks the received formula for consistency.

**Returns**

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.64.4.21 `checkInfeasibleSubsetForMinimality()`** `void smtrat::Module::checkInfeasibleSubsetForMinimality (`  
`std::vector< FormulaSetT >::const_iterator _infsSubset,`  
`const std::string & _filename = "smaller_muses",`  
`unsigned _maxSizeDifference = 1 ) const [inherited]`

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.64.4.22 `checkModel()`** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.64.4.23 `cleanModel()`** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.64.4.24 `clearLemmas()`** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

**0.15.64.4.25 `clearModel()`** `void smtrat::Module::clearModel () const [inline], [protected], [inherited]`  
 Clears the assignment, if any was found.

**0.15.64.4.26 `clearModels()`** `void smtrat::Module::clearModels () const [inline], [protected], [inherited]`  
 Clears all assignments, if any was found.

**0.15.64.4.27 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula () [protected], [inherited]`

**0.15.64.4.28 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins (const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.64.4.29 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins (const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.64.4.30 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations () [inherited]`

**0.15.64.4.31 `constraintsToInform()`** `const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]`

#### Returns

The constraints which the backends must be informed about.

**0.15.64.4.32 `currentlySatisfied()`** `virtual unsigned smtrat::Module::currentlySatisfied (const FormulaT & ) const [inline], [virtual], [inherited]`

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

```
0.15.64.4.33 currentlySatisfiedByBackend() unsigned smrat::Module::currentlySatisfiedByBackend
(
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.64.4.34 deinform() void smrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

```
0.15.64.4.35 deinformCore() virtual void smrat::Module::deinformCore (
 const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), and [smrat::LRAModule< LRASetting >](#).

```
0.15.64.4.36 determine_smallest_origin() size_t smrat::Module::determine_smallest_origin (
 const std::vector< FormulaT > & origins) const [protected], [inherited]
```

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

```
0.15.64.4.37 eraseSubformulaFromPassedFormula() ModuleInput::iterator smrat::Module::erase←
SubformulaFromPassedFormula (
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to remove from the passed formula. |
|--------------------------|----------------------------------------------------|

## Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|

## Returns

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.64.4.38 `evaluateBVFormula()`** `template<typename Settings >`  
`size_t smtrat::BVModule< Settings >::evaluateBVFormula (`  
`const FormulaT & formula ) [protected]`

**0.15.64.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`  
Excludes all variables from the current model, which do not occur in the received formula.

**0.15.64.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`  
`const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

## Parameters

|                      |  |
|----------------------|--|
| <code>origins</code> |  |
|----------------------|--|

## Returns

**0.15.64.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.64.4.42 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

## Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.64.4.43 `freeSplittingVariable()`** `static void smtrat::Module::freeSplittingVariable (`  
`const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.64.4.44 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.64.4.45 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.64.4.46 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.64.4.47 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.64.4.48 getInfeasibleSubsets() [2/2]** std::vector< [FormulaSetT](#) > smtrat::Module::getInfeasibleSubsets ( )  
const [Module](#) & \_backend ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.64.4.49 getInformationRelevantFormulas()** const std::set< [FormulaT](#) > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.64.4.50 getModelEqualities()** std::list< std::vector< [carl::Variable](#) > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.64.4.51 getOrigins() [1/3]** void smtrat::Module::getOrigins ( )  
const [FormulaT](#) & \_formula,  
[FormulaSetT](#) & \_origins ) const [inline], [protected], [inherited]

## Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.64.4.52 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins (`  
`const FormulaT & _formula,`  
`FormulasT & _origins ) const [inline], [protected], [inherited]`

## Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.64.4.53 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins (`  
`ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

## Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

## Returns

The origins of the passed formula at the given position.

**0.15.64.4.54 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::Module::get←`  
`ReceivedFormulaSimplified ( ) [virtual], [inherited]`

## Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.64.4.55 `hasLemmas()`** `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.64.4.56 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset ( ) const`  
`[inherited]`

## Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.64.4.57 `id()`** `std::size_t smrat::Module::id( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.64.4.58 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smrat::Module::infeasibleSubsets( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.64.4.59 `inform()`** `bool smrat::Module::inform( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.64.4.60 `informBackends()`** `void smrat::Module::informBackends( const FormulaT & _constraint ) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.64.4.61 `informCore()`** `template<typename Settings > bool smrat::BVModule< Settings >::informCore( const FormulaT & _constraint ) [virtual]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.64.4.62 informedConstraints()** const carl::FastSet<[FormulaT](#)>& smrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.64.4.63 init()** template<typename Settings >  
void smrat::BVModule< Settings >::init ( ) [virtual]

Informs all backends about the so far encountered constraints, which have not yet been communicated.  
This method must not and will not be called more than once and only before the first runBackends call.  
Reimplemented from [smrat::Module](#).

**0.15.64.4.64 is\_minimizing()** bool smrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.64.4.65 isLemmaLevel()** bool smrat::Module::isLemmaLevel ( [LemmaLevel](#) level ) [protected], [inherited]

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.64.4.66 isPreprocessor()** bool smrat::Module::isPreprocessor ( ) const [inline], [inherited]

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.64.4.67 lemmas()** const std::vector<[Lemma](#)>& smrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.64.4.68 merge()** std::vector< [FormulaT](#) > smrat::Module::merge ( const std::vector< [FormulaT](#) > & \_vecSetA, const std::vector< [FormulaT](#) > & \_vecSetB ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.64.4.69 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.64.4.70 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (`

```
 const Model & _modelA,
 const Model & _modelB) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.64.4.71 `moduleName()`** `template<typename Settings >  
std::string smtrat::BVMModule< Settings >::moduleName () const [inline], [virtual]`**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.64.4.72 `objective()`** `carl::Variable smtrat::Module::objective () const [inline], [inherited]`**0.15.64.4.73 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`

```
 const FormulaT & _origin) const [protected], [inherited]
```

**0.15.64.4.74 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin () [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.64.4.75 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.64.4.76 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.64.4.77 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.64.4.78 print()** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const [inherited]`  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.64.4.79 printAllModels()** `void smtrat::Module::printAllModels (`  
`std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.64.4.80 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets (`  
`const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.64.4.81 printModel()** `void smtrat::Module::printModel (`  
`std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.64.4.82 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.64.4.83 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.64.4.84 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.64.4.85 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.64.4.86 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAs<-InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

```
0.15.64.4.87 receivedVariable() bool smrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.64.4.88 remove() void smrat::Module::remove (
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub formula of the received formula to remove. |
|-------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

```
0.15.64.4.89 removeCore() template<typename Settings >
void smrat::BVModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|             |                                           |
|-------------|-------------------------------------------|
| _subformula | The position of the subformula to remove. |
|-------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

```
0.15.64.4.90 removeOrigin() std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

```
0.15.64.4.91 removeOrigins() std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins (
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

```
0.15.64.4.92 rPassedFormula() const ModuleInput& smrat::Module::rPassedFormula () const
[inline], [inherited]
```

#### Returns

A reference to the formula passed to the backends of this module.

```
0.15.64.4.93 rReceivedFormula() const ModuleInput& smrat::Module::rReceivedFormula () const
[inline], [inherited]
```

**Returns**

A reference to the formula passed to this module.

**0.15.64.4.94 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.64.4.95 runBackends() [2/2]** `Answer smtrat::Module::runBackends ( bool _final, bool _full, carl::Variable _objective ) [protected], [virtual], [inherited]`

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.64.4.96 setId()** `void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

**Parameters**

|                   |                                    |
|-------------------|------------------------------------|
| <code>↔ id</code> | The id to set this module's id to. |
|-------------------|------------------------------------|

**0.15.64.4.97 setThreadPriority()** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.64.4.98 solverState()** `Answer smtrat::Module::solverState () const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.64.4.99 splitUnequalConstraint()** `void smrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.64.4.100 threadPriority()** `thread_priority smrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.64.4.101 transferBackendModel()** `template<typename Settings >`

`void smrat::BVModule< Settings >::transferBackendModel ( ) const [protected]`

**0.15.64.4.102 updateAllModels()** `void smrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in `smrat::SATModule< Settings >`.

**0.15.64.4.103 updateLemmas()** `void smrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.64.4.104 updateModel()** `template<typename Settings >`

`void smrat::BVModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from `smrat::Module`.

**0.15.64.4.105 usedBackends()** `const std::vector<Module*>& smrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.64.5 Field Documentation**

**0.15.64.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends` [protected], [inherited]

The backends of this module which have been used.

**0.15.64.5.2 mAllModels** `std::vector<Model> smtrat::Module::mAllModels` [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.64.5.3 mBackendsFoundAnswer** `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer` [protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.64.5.4 mConstraintsToInform** `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform` [protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.64.5.5 mFinalCheck** `bool smtrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.64.5.6 mFirstPosInLastBranches** `std::size_t smtrat::Module::mFirstPosInLastBranches = 0` [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.64.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass` [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.64.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.64.5.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.64.5.10 mFullCheck** `bool smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.64.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets` [protected], [inherited]

Stores the infeasible subsets.

**0.15.64.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.64.5.13 mLastBranches** `std::vector< Branching > smtrat::Module::mLastBranches = std::vector< Branching > (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.64.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.64.5.15 mModel Model** `smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.64.5.16 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.64.5.17 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.64.5.18 mOldSplittingVariables** `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.64.5.19 mpManager Manager\*** `const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.64.5.20 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter` [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.64.5.21 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.64.5.22 mTheoryPropagations** `std::vector< TheoryPropagation > smtrat::Module::mTheoryPropagations` [protected], [inherited]

**0.15.64.5.23 mUsedBackends** `std::vector< Module* > smtrat::Module::mUsedBackends` [protected], [inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.64.5.24 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters`  
[protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.65 `smtrat::by_address_hasher<IterType>` Struct Template Reference

Hash an iterator by the pointed-to address.

```
#include <IterHash.h>
```

### Public Types

- `typedef std::iterator_traits<IterType>::pointer pointer_type`

### Public Member Functions

- `std::size_t operator()(const IterType &iter) const noexcept`

#### 0.15.65.1 Detailed Description

```
template<typename IterType>
struct smtrat::by_address_hasher<IterType>
```

Hash an iterator by the pointed-to address.

#### 0.15.65.2 Member Typedef Documentation

**0.15.65.2.1 pointer\_type** `template<typename IterType>`  
`typedef std::iterator_traits<IterType>::pointer smtrat::by_address_hasher<IterType>::pointer_type`

#### 0.15.65.3 Member Function Documentation

**0.15.65.3.1 operator()** `template<typename IterType>`  
`std::size_t smtrat::by_address_hasher<IterType>::operator()(const IterType & iter) const [inline], [noexcept]`

## 0.15.66 `smtrat::cad::CAD<Settings>` Class Template Reference

```
#include <CAD.h>
```

### Public Types

- `using SettingsT = Settings`

### Public Member Functions

- `CAD()`
- `~CAD()`
- `std::size_t dim() const`
- `const auto & getVariables() const`
- `const auto & getProjection() const`
- `const auto & getLifting() const`
- `const auto & getConstraints() const`
- `const auto & getConstraintMap() const`
- `bool isIdValid(std::size_t id) const`
- `const auto & getBounds() const`

- void `reset` (const std::vector< carl::Variable > &vars)
- void `addConstraint` (const `ConstraintT` &c)
- void `removeConstraint` (const `ConstraintT` &c)
- template<typename ConstraintT>  
  bool `evaluateSample` (Sample &sample, const ConstraintIt &constraint, Assignment &assignment) const
- std::vector< Assignment > `enumerateSolutions` ()
- Answer `checkFullSamples` (Assignment &assignment)
- template<typename Iterator>  
  Answer `checkPartialSample` (Iterator &it, std::size\_t level)
- Answer `check` (Assignment &assignment, std::vector< `FormulaSetT` > &mis)
- ConflictGraph `generateConflictGraph` () const
- auto `generateCovering` () const

## Data Fields

- debug::TikzHistoryPrinter `thp`

## Friends

- template<CoreHeuristic CH>  
  struct `CADCore`

### 0.15.66.1 Member Typedef Documentation

**0.15.66.1.1 SettingsT** template<typename Settings >  
using `smtrat::cad::CAD< Settings >::SettingsT` = `Settings`

### 0.15.66.2 Constructor & Destructor Documentation

**0.15.66.2.1 CAD()** template<typename Settings >  
`smtrat::cad::CAD< Settings >::CAD` ( ) [inline]

**0.15.66.2.2 ~CAD()** template<typename Settings >  
`smtrat::cad::CAD< Settings >::~CAD` ( ) [inline]

### 0.15.66.3 Member Function Documentation

**0.15.66.3.1 addConstraint()** template<typename Settings >  
void `smtrat::cad::CAD< Settings >::addConstraint` (  
  const `ConstraintT` & c ) [inline]

**0.15.66.3.2 check()** template<typename Settings >  
Answer `smtrat::cad::CAD< Settings >::check` (  
  Assignment & assignment,  
  std::vector< `FormulaSetT` > & mis ) [inline]

```
0.15.66.3.3 checkFullSamples() template<typename Settings >
Answer smtrat::cad::CAD< Settings >::checkFullSamples (
 Assignment & assignment) [inline]
```

```
0.15.66.3.4 checkPartialSample() template<typename Settings >
template<typename Iterator >
Answer smtrat::cad::CAD< Settings >::checkPartialSample (
 Iterator & it,
 std::size_t level) [inline]
```

```
0.15.66.3.5 dim() template<typename Settings >
std::size_t smtrat::cad::CAD< Settings >::dim () const [inline]
```

```
0.15.66.3.6 enumerateSolutions() template<typename Settings >
std::vector<Assignment> smtrat::cad::CAD< Settings >::enumerateSolutions () [inline]
```

```
0.15.66.3.7 evaluateSample() template<typename Settings >
template<typename ConstraintIt >
bool smtrat::cad::CAD< Settings >::evaluateSample (
 Sample & sample,
 const ConstraintIt & constraint,
 Assignment & assignment) const [inline]
```

```
0.15.66.3.8 generateConflictGraph() template<typename Settings >
ConflictGraph smtrat::cad::CAD< Settings >::generateConflictGraph () const [inline]
```

```
0.15.66.3.9 generateCovering() template<typename Settings >
auto smtrat::cad::CAD< Settings >::generateCovering () const [inline]
```

```
0.15.66.3.10 getBounds() template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getBounds () const [inline]
```

```
0.15.66.3.11 getConstraintMap() template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getConstraintMap () const [inline]
```

```
0.15.66.3.12 getConstraints() template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getConstraints () const [inline]
```

```
0.15.66.3.13 getLifting() template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getLifting () const [inline]
```

```
0.15.66.3.14 getProjection() template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getProjection () const [inline]
```

```
0.15.66.3.15 getVariables() template<typename Settings >
const auto& smtrat::cad::CAD< Settings >::getVariables () const [inline]
```

```
0.15.66.3.16 isIdValid() template<typename Settings >
bool smtrat::cad::CAD< Settings >::isIdValid (
 std::size_t id) const [inline]
```

```
0.15.66.3.17 removeConstraint() template<typename Settings >
void smtrat::cad::CAD< Settings >::removeConstraint (
 const ConstraintT & c) [inline]
```

```
0.15.66.3.18 reset() template<typename Settings >
void smtrat::cad::CAD< Settings >::reset (
 const std::vector< carl::Variable > & vars) [inline]
```

## 0.15.66.4 Friends And Related Function Documentation

```
0.15.66.4.1 CADCore template<typename Settings >
template<CoreHeuristic CH>
friend struct CADCore [friend]
```

## 0.15.66.5 Field Documentation

```
0.15.66.5.1 thp template<typename Settings >
debug::TikzHistoryPrinter smtrat::cad::CAD< Settings >::thp
```

## 0.15.67 smtrat::qe::cad::CAD< Settings > Class Template Reference

```
#include <CAD.h>
```

### Public Types

- using Treeliterator = typename smtrat::cad::LiftingTree< Settings >::Iterator

### Public Member Functions

- CAD ()
- void reset (const std::vector< carl::Variable > &vars)
- std::size\_t dim () const
- const auto & getProjection () const
- const auto & getLifting () const
- void addConstraint (const ConstraintT &c)
- void removeConstraint (const ConstraintT &c)
- void addPolynomial (const Poly &p)
- void removePolynomial (std::size\_t level, std::size\_t id)
- void project ()
- void lift ()

### 0.15.67.1 Member Typedef Documentation

```
0.15.67.1.1 Treeliterator template<typename Settings >
using smtrat::qe::cad::CAD< Settings >::TreeIterator = typename smtrat::cad::LiftingTree<Settings>::Iterator
```

## 0.15.67.2 Constructor & Destructor Documentation

```
0.15.67.2.1 CAD() template<typename Settings >
smtrat::qe::cad::CAD< Settings >::CAD () [inline]
```

## 0.15.67.3 Member Function Documentation

```
0.15.67.3.1 addConstraint() template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::addConstraint (
 const ConstraintT & c) [inline]
```

```
0.15.67.3.2 addPolynomial() template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::addPolynomial (
 const Poly & p) [inline]
```

```
0.15.67.3.3 dim() template<typename Settings >
std::size_t smtrat::qe::cad::CAD< Settings >::dim () const [inline]
```

```
0.15.67.3.4 getLifting() template<typename Settings >
const auto& smtrat::qe::cad::CAD< Settings >::getLifting () const [inline]
```

```
0.15.67.3.5 getProjection() template<typename Settings >
const auto& smtrat::qe::cad::CAD< Settings >::getProjection () const [inline]
```

```
0.15.67.3.6 lift() template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::lift () [inline]
```

```
0.15.67.3.7 project() template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::project () [inline]
```

```
0.15.67.3.8 removeConstraint() template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::removeConstraint (
 const ConstraintT & c) [inline]
```

```
0.15.67.3.9 removePolynomial() template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::removePolynomial (
 std::size_t level,
 std::size_t id) [inline]
```

```
0.15.67.3.10 reset() template<typename Settings >
void smtrat::qe::cad::CAD< Settings >::reset (
 const std::vector< carl::Variable > & vars) [inline]
```

## 0.15.68 smtrat::cad::CADConstraints< BT > Class Template Reference

```
#include <CADConstraints.h>
```

### Data Structures

- struct [ConstraintComparator](#)

### Public Types

- using [Callback](#) = std::function< void(const [UPoly](#) &, std::size\_t, bool)>
- using [VariableBounds](#) = [vb::VariableBounds](#)< [ConstraintT](#) >

### Public Member Functions

- [CADConstraints](#) (const [Callback](#) &onAdd, const [Callback](#) &onAddEq, const [Callback](#) &onRemove)
- [CADConstraints](#) (const [CADConstraints](#) &) = delete
- void [reset](#) (const std::vector< carl::Variable > & vars)
- const std::vector< carl::Variable > & [vars](#) () const
- std::size\_t [size](#) () const
- bool [valid](#) (std::size\_t id) const
- const auto & [indexed](#) () const
- const auto & [ordered](#) () const
- const auto & [bounds](#) () const
- const auto & [unsatByBounds](#) () const
- std::size\_t [add](#) (const [ConstraintT](#) &c)
- carl::Bitset [remove](#) (const [ConstraintT](#) &c)
   
*Removes a constraint.*
- const [ConstraintT](#) & [operator\[\]](#) (std::size\_t id) const
- std::size\_t [level](#) (std::size\_t id) const
- bool [checkForTrivialConflict](#) (std::vector< [FormulaSetT](#) > & mis) const
- void [exportAsDot](#) (std::ostream &out) const

### Protected Types

- using [ConstraintMap](#) = std::map< [ConstraintT](#), std::size\_t, [ConstraintComparator](#) >
- using [ConstraintItrs](#) = std::vector< typename [ConstraintMap](#)::iterator >

### Protected Member Functions

- template<typename CB , typename... Args>
 void [callCallback](#) (const CB &cb, const [ConstraintT](#) &c, Args... args) const

### Protected Attributes

- std::vector< carl::Variable > [mVariables](#)
- [Callback](#) [mAddCallback](#)
- [Callback](#) [mAddEqCallback](#)
- [Callback](#) [mRemoveCallback](#)
- [ConstraintMap](#) [mActiveConstraintMap](#)
- [ConstraintMap](#) [mConstraintMap](#)
- std::vector< typename [ConstraintMap](#)::iterator > [mConstraintItrs](#)
- std::vector< std::size\_t > [mConstraintLevels](#)

- carl::IDPool `mIDPool`
- `VariableBounds mBounds`
- carl::Bitset `mSatByBounds`  
*List of constraints that are satisfied by bounds.*
- carl::Bitset `mUnsatByBounds`  
*List of constraints that are infeasible due to bounds.*

## Friends

- template<Backtracking B>  
  `std::ostream & operator<< (std::ostream &os, const CADConstraints< B > &cc)`

### 0.15.68.1 Member Typedef Documentation

**0.15.68.1.1 Callback** template<Backtracking BT>  
using `smtrat::cad::CADConstraints< BT >::Callback` = std::function<void(const `UPoly`&, std::size\_t, bool)>

**0.15.68.1.2 ConstraintIts** template<Backtracking BT>  
using `smtrat::cad::CADConstraints< BT >::ConstraintIts` = std::vector<typename `ConstraintMap`::iterator> [protected]

**0.15.68.1.3 ConstraintMap** template<Backtracking BT>  
using `smtrat::cad::CADConstraints< BT >::ConstraintMap` = std::map<`ConstraintT`, std::size\_t, `ConstraintComparator`> [protected]

**0.15.68.1.4 VariableBounds** template<Backtracking BT>  
using `smtrat::cad::CADConstraints< BT >::VariableBounds` = `vb::VariableBounds<ConstraintT>`

### 0.15.68.2 Constructor & Destructor Documentation

**0.15.68.2.1 CADConstraints() [1/2]** template<Backtracking BT>  
`smtrat::cad::CADConstraints< BT >::CADConstraints` (  
  const `Callback` & `onAdd`,  
  const `Callback` & `onAddEq`,  
  const `Callback` & `onRemove`) [inline]

**0.15.68.2.2 CADConstraints() [2/2]** template<Backtracking BT>  
`smtrat::cad::CADConstraints< BT >::CADConstraints` (  
  const `CADConstraints< BT >` & ) [delete]

### 0.15.68.3 Member Function Documentation

**0.15.68.3.1 add()** template<Backtracking BT>  
`std::size_t smtrat::cad::CADConstraints< BT >::add` (  
  const `ConstraintT` & `c`) [inline]

```
0.15.68.3.2 bounds() template<Backtracking BT>
const auto& smtrat::cad::CADConstraints< BT >::bounds () const [inline]
```

```
0.15.68.3.3 callCallback() template<Backtracking BT>
template<typename CB , typename... Args>
void smtrat::cad::CADConstraints< BT >::callCallback (
 const CB & cb,
 const ConstraintT & c,
 Args... args) const [inline], [protected]
```

```
0.15.68.3.4 checkForTrivialConflict() template<Backtracking BT>
bool smtrat::cad::CADConstraints< BT >::checkForTrivialConflict (
 std::vector< FormulaSetT > & mis) const [inline]
```

```
0.15.68.3.5 exportAsDot() template<Backtracking BT>
void smtrat::cad::CADConstraints< BT >::exportAsDot (
 std::ostream & out) const [inline]
```

```
0.15.68.3.6 indexed() template<Backtracking BT>
const auto& smtrat::cad::CADConstraints< BT >::indexed () const [inline]
```

```
0.15.68.3.7 level() template<Backtracking BT>
std::size_t smtrat::cad::CADConstraints< BT >::level (
 std::size_t id) const [inline]
```

```
0.15.68.3.8 operator[]() template<Backtracking BT>
const ConstraintT& smtrat::cad::CADConstraints< BT >::operator[] (
 std::size_t id) const [inline]
```

```
0.15.68.3.9 ordered() template<Backtracking BT>
const auto& smtrat::cad::CADConstraints< BT >::ordered () const [inline]
```

```
0.15.68.3.10 remove() template<Backtracking BT>
carl::Bitset smtrat::cad::CADConstraints< BT >::remove (
 const ConstraintT & c) [inline]
```

Removes a constraint.

Returns the set of constraint ids that have (possibly) been reassigned and should be cleared from the sample evaluations.

```
0.15.68.3.11 reset() template<Backtracking BT>
void smtrat::cad::CADConstraints< BT >::reset (
 const std::vector< carl::Variable > & vars) [inline]
```

```
0.15.68.3.12 size() template<Backtracking BT>
std::size_t smtrat::cad::CADConstraints< BT >::size () const [inline]
```

**0.15.68.3.13 `unsatByBounds()`** template<Backtracking BT>  
const auto& smtrat::cad::CADConstraints< BT >::unsatByBounds ( ) const [inline]

**0.15.68.3.14 `valid()`** template<Backtracking BT>  
bool smtrat::cad::CADConstraints< BT >::valid ( std::size\_t id ) const [inline]

**0.15.68.3.15 `vars()`** template<Backtracking BT>  
const std::vector<carl::Variable>& smtrat::cad::CADConstraints< BT >::vars ( ) const [inline]

#### 0.15.68.4 Friends And Related Function Documentation

**0.15.68.4.1 `operator<<`** template<Backtracking BT>  
template<Backtracking B>  
std::ostream& operator<< ( std::ostream & os, const CADConstraints< B > & cc ) [friend]

#### 0.15.68.5 Field Documentation

**0.15.68.5.1 `mActiveConstraintMap`** template<Backtracking BT>  
ConstraintMap smtrat::cad::CADConstraints< BT >::mActiveConstraintMap [protected]

**0.15.68.5.2 `mAddCallback`** template<Backtracking BT>  
Callback smtrat::cad::CADConstraints< BT >::mAddCallback [protected]

**0.15.68.5.3 `mAddEqCallback`** template<Backtracking BT>  
Callback smtrat::cad::CADConstraints< BT >::mAddEqCallback [protected]

**0.15.68.5.4 `mBounds`** template<Backtracking BT>  
VariableBounds smtrat::cad::CADConstraints< BT >::mBounds [protected]

**0.15.68.5.5 `mConstraintIts`** template<Backtracking BT>  
std::vector<typename ConstraintMap::iterator> smtrat::cad::CADConstraints< BT >::mConstraintIts [protected]

**0.15.68.5.6 `mConstraintLevels`** template<Backtracking BT>  
std::vector<std::size\_t> smtrat::cad::CADConstraints< BT >::mConstraintLevels [protected]

**0.15.68.5.7 `mConstraintMap`** template<Backtracking BT>  
ConstraintMap smtrat::cad::CADConstraints< BT >::mConstraintMap [protected]

**0.15.68.5.8 mIDPool** template<Backtracking BT>  
carl::IDPool smtrat::cad::CADConstraints< BT >::mIDPool [protected]

**0.15.68.5.9 mRemoveCallback** template<Backtracking BT>  
Callback smtrat::cad::CADConstraints< BT >::mRemoveCallback [protected]

**0.15.68.5.10 mSatByBounds** template<Backtracking BT>  
carl::Bitset smtrat::cad::CADConstraints< BT >::mSatByBounds [protected]  
List of constraints that are satisfied by bounds.

**0.15.68.5.11 mUnsatByBounds** template<Backtracking BT>  
carl::Bitset smtrat::cad::CADConstraints< BT >::mUnsatByBounds [protected]  
List of constraints that are infeasible due to bounds.

**0.15.68.5.12 mVariables** template<Backtracking BT>  
std::vector<carl::Variable> smtrat::cad::CADConstraints< BT >::mVariables [protected]

## 0.15.69 smtrat::cad::CADCore< CH > Struct Template Reference

#include <CADCore.h>

## 0.15.70 smtrat::cad::CADCore< CoreHeuristic::BySample > Struct Reference

#include <CADCore.h>

### Public Member Functions

- template<typename CAD >  
[Answer operator\(\)](#) ([Assignment](#) &assignment, [CAD](#) &cad)

#### 0.15.70.1 Member Function Documentation

**0.15.70.1.1 operator()** template<typename CAD >  
Answer smtrat::cad::CADCore< CoreHeuristic::BySample >::operator() (  
    Assignment & assignment,  
    CAD & cad ) [inline]

## 0.15.71 smtrat::cad::CADCore< CoreHeuristic::EnumerateAll > Struct Reference

#include <CADCore.h>

### Public Member Functions

- template<typename CAD >  
[Answer operator\(\)](#) ([Assignment](#) &assignment, [CAD](#) &cad)

#### 0.15.71.1 Member Function Documentation

```
0.15.71.1.1 operator() template<typename CAD >
Answer smrat::cad::CADCore< CoreHeuristic::EnumerateAll >::operator() (
 Assignment & assignment,
 CAD & cad) [inline]
```

## 0.15.72 smrat::cad::CADCore< CoreHeuristic::Interleave > Struct Reference

```
#include <CADCore.h>
```

### Public Member Functions

- template<typename It >  
  bool preferLifting (const It &it)
- template<typename CAD >  
  bool doProjection (CAD &cad)
- template<typename CAD >  
  bool doLifting (CAD &cad)
- template<typename CAD >  
  Answer operator() (Assignment &assignment, CAD &cad)

### 0.15.72.1 Member Function Documentation

```
0.15.72.1.1 doLifting() template<typename CAD >
bool smrat::cad::CADCore< CoreHeuristic::Interleave >::doLifting (
 CAD & cad) [inline]
```

```
0.15.72.1.2 doProjection() template<typename CAD >
bool smrat::cad::CADCore< CoreHeuristic::Interleave >::doProjection (
 CAD & cad) [inline]
```

```
0.15.72.1.3 operator() template<typename CAD >
Answer smrat::cad::CADCore< CoreHeuristic::Interleave >::operator() (
 Assignment & assignment,
 CAD & cad) [inline]
```

```
0.15.72.1.4 preferLifting() template<typename It >
bool smrat::cad::CADCore< CoreHeuristic::Interleave >::preferLifting (
 const It & it) [inline]
```

## 0.15.73 smrat::cad::CADCore< CoreHeuristic::PreferProjection > Struct Reference

```
#include <CADCore.h>
```

### Public Member Functions

- template<typename CAD >  
  Answer operator() (Assignment &assignment, CAD &cad)

### 0.15.73.1 Member Function Documentation

```
0.15.73.1.1 operator() template<typename CAD >
Answer smtrat::cad::CADCore< CoreHeuristic::PreferProjection >::operator() (
 Assignment & assignment,
 CAD & cad) [inline]
```

## 0.15.74 smtrat::cad::CADCore< CoreHeuristic::PreferSampling > Struct Reference

```
#include <CADCore.h>
```

### Public Member Functions

- template<typename CAD >  
Answer operator() (Assignment &assignment, CAD &cad)

#### 0.15.74.1 Member Function Documentation

```
0.15.74.1.1 operator() template<typename CAD >
Answer smtrat::cad::CADCore< CoreHeuristic::PreferSampling >::operator() (
 Assignment & assignment,
 CAD & cad) [inline]
```

## 0.15.75 smtrat::qe::cad::CADElimination Class Reference

```
#include <CADElimination.h>
```

### Public Member Functions

- CADElimination (const FormulaT &quantifierFreePart, const QEQuery &quantifiers)
- FormulaT eliminateQuantifiers ()

#### 0.15.75.1 Constructor & Destructor Documentation

```
0.15.75.1.1 CADElimination() smtrat::qe::cad::CADElimination::CADElimination (
 const FormulaT & quantifierFreePart,
 const QEQuery & quantifiers)
```

#### 0.15.75.2 Member Function Documentation

```
0.15.75.2.1 eliminateQuantifiers() FormulaT smtrat::qe::cad::CADElimination::eliminateQuantifiers
()
```

## 0.15.76 smtrat::cad::CADPreprocessor Class Reference

```
#include <CADPreprocessor.h>
```

### Public Member Functions

- CADPreprocessor (const std::vector< carl::Variable > &vars)
- const auto & equalities () const
- const auto & inequalities () const
- void addConstraint (const ConstraintT &c)
- void removeConstraint (const ConstraintT &c)

- `bool preprocess ()`  
*Performs the preprocessing.*
- `const Model & model () const`
- `template<typename Map>`  
`preprocessor::ConstraintUpdate result (const Map &oldC) const`
- `std::set< FormulaT > getConflict () const`
- `void postprocessConflict (std::set< FormulaT > &mis) const`
- `FormulaT simplify (const FormulaT &f) const`

## Friends

- `std::ostream & operator<< (std::ostream &os, const CADPreprocessor &cadpp)`

### 0.15.76.1 Constructor & Destructor Documentation

**0.15.76.1.1 CADPreprocessor()** `smtrat::cad::CADPreprocessor::CADPreprocessor (`  
`const std::vector< carl::Variable > & vars ) [inline]`

### 0.15.76.2 Member Function Documentation

**0.15.76.2.1 addConstraint()** `void smtrat::cad::CADPreprocessor::addConstraint (`  
`const ConstraintT & c )`

**0.15.76.2.2 equalities()** `const auto& smtrat::cad::CADPreprocessor::equalities () const [inline]`

**0.15.76.2.3 getConflict()** `std::set< FormulaT > smtrat::cad::CADPreprocessor::getConflict () const`

**0.15.76.2.4 inequalities()** `const auto& smtrat::cad::CADPreprocessor::inequalities () const [inline]`

**0.15.76.2.5 model()** `const Model& smtrat::cad::CADPreprocessor::model () const [inline]`

**0.15.76.2.6 postprocessConflict()** `void smtrat::cad::CADPreprocessor::postprocessConflict (`  
`std::set< FormulaT > & mis ) const`

**0.15.76.2.7 preprocess()** `bool smtrat::cad::CADPreprocessor::preprocess ()`

Performs the preprocessing.

Return false if a direct conflict was found, true otherwise.

**0.15.76.2.8 removeConstraint()** `void smtrat::cad::CADPreprocessor::removeConstraint (`  
`const ConstraintT & c )`

```
0.15.76.2.9 result() template<typename Map >
preprocessor::ConstraintUpdate smtrat::cad::CADPreprocessor::result (
 const Map & oldC) const [inline]
```

```
0.15.76.2.10 simplify() FormulaT smtrat::cad::CADPreprocessor::simplify (
 const FormulaT & f) const [inline]
```

### 0.15.76.3 Friends And Related Function Documentation

```
0.15.76.3.1 operator<< std::ostream& operator<< (
 std::ostream & os,
 const CADPreprocessor & cadpp) [friend]
```

## 0.15.77 smtrat::cad::CADPreprocessorSettings Struct Reference

```
#include <CADPreprocessor.h>
```

### Static Public Member Functions

- static void **register\_settings** (SettingsParser &parser)
- static bool **register\_hook** ()

### Data Fields

- bool **disable\_variable\_elimination** = false
- bool **disable\_resultants** = false

### Static Public Attributes

- static const bool **dummy** = CADPreprocessorSettings::register\_hook()

#### 0.15.77.1 Member Function Documentation

```
0.15.77.1.1 register_hook() static bool smtrat::cad::CADPreprocessorSettings::register_hook ()
[inline], [static]
```

```
0.15.77.1.2 register_settings() static void smtrat::cad::CADPreprocessorSettings::register_←
settings (
 SettingsParser & parser) [inline], [static]
```

#### 0.15.77.2 Field Documentation

```
0.15.77.2.1 disable_resultants bool smtrat::cad::CADPreprocessorSettings::disable_resultants =
false
```

```
0.15.77.2.2 disable_variable_elimination bool smtrat::cad::CADPreprocessorSettings::disable_←
variable_elimination = false
```

**0.15.77.2.3 dummy** const bool smtrat::cad::CADPreprocessorSettings::dummy = `CADPreprocessorSettings::register`  
[static]

## 0.15.78 smtrat::qe::cad::CADSettings Struct Reference

```
#include <CADElimination.h>
```

### Static Public Attributes

- static constexpr smtrat::cad::ProjectionType projectionOperator = smtrat::cad::ProjectionType::Brown
- static constexpr smtrat::cad::Incrementality incrementality = smtrat::cad::Incrementality::NONE
- static constexpr smtrat::cad::Backtracking backtracking = smtrat::cad::Backtracking::UNORDERED
- static constexpr CoreHeuristic coreHeuristic = cad::CoreHeuristic::PreferProjection
- static constexpr MISHeuristic misHeuristic = cad::MISHeuristic::GREEDY
- static constexpr std::size\_t trivialSampleRadius = 1
- static constexpr bool simplifyProjectionByBounds = true
- static constexpr ProjectionCompareStrategy projectionComparator = cad::ProjectionCompareStrategy::Default
- static constexpr SampleCompareStrategy sampleComparator = cad::SampleCompareStrategy::Default
- static constexpr FullSampleCompareStrategy fullSampleComparator = cad::FullSampleCompareStrategy::Default

### 0.15.78.1 Field Documentation

**0.15.78.1.1 backtracking** constexpr smtrat::cad::Backtracking smtrat::qe::cad::CADSettings::backtracking = `smtrat::cad::Backtracking::UNORDERED` [static], [constexpr]

**0.15.78.1.2 coreHeuristic** constexpr CoreHeuristic smtrat::cad::BaseSettings::coreHeuristic = `cad::CoreHeuristic::PreferProjection` [static], [constexpr], [inherited]

**0.15.78.1.3 fullSampleComparator** constexpr FullSampleCompareStrategy smtrat::cad::BaseSettings::fullSampleComparator = `cad::FullSampleCompareStrategy::Default` [static], [constexpr], [inherited]

**0.15.78.1.4 incrementality** constexpr smtrat::cad::Incrementality smtrat::qe::cad::CADSettings::incrementality = `smtrat::cad::Incrementality::NONE` [static], [constexpr]

**0.15.78.1.5 misHeuristic** constexpr MISHeuristic smtrat::cad::BaseSettings::misHeuristic = `cad::MISHeuristic::GREEDY` [static], [constexpr], [inherited]

**0.15.78.1.6 projectionComparator** constexpr ProjectionCompareStrategy smtrat::cad::BaseSettings::projectionComparator = `cad::ProjectionCompareStrategy::Default` [static], [constexpr], [inherited]

**0.15.78.1.7 projectionOperator** constexpr smtrat::cad::ProjectionType smtrat::qe::cad::CADSettings::projectionOperator = `smtrat::cad::ProjectionType::Brown` [static], [constexpr]

**0.15.78.1.8 sampleComparator** constexpr SampleCompareStrategy smtrat::cad::BaseSettings::sampleComparator = `cad::SampleCompareStrategy::Default` [static], [constexpr], [inherited]

---

**0.15.78.1.9 `simplifyProjectionByBounds`** `constexpr bool smtrat::cad::BaseSettings::simplify←ProjectionByBounds = true [static], [constexpr], [inherited]`

**0.15.78.1.10 `trivialSampleRadius`** `constexpr std::size_t smtrat::cad::BaseSettings::trivial←SampleRadius = 1 [static], [constexpr], [inherited]`

## 0.15.79 `smtrat::CardinalityEncoder` Class Reference

```
#include <CardinalityEncoder.h>
```

### Public Member Functions

- `CardinalityEncoder ()`
- `Rational encodingSize (const ConstraintT &constraint)`
- `bool canEncode (const ConstraintT &constraint)`
- `std::string name ()`
- `std::optional< FormulaT > encode (const ConstraintT &constraint)`

*Encodes an arbitrary constraint.*

### Data Fields

- `std::size_t problem_size`

### Protected Member Functions

- `std::optional< FormulaT > doEncode (const ConstraintT &constraint)`
- `FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)`

#### 0.15.79.1 Constructor & Destructor Documentation

**0.15.79.1.1 `CardinalityEncoder()`** `smtrat::CardinalityEncoder::CardinalityEncoder () [inline]`

#### 0.15.79.2 Member Function Documentation

**0.15.79.2.1 `canEncode()`** `bool smtrat::CardinalityEncoder::canEncode ( const ConstraintT & constraint ) [virtual]`

Implements `smtrat::PseudoBoolEncoder`.

**0.15.79.2.2 `doEncode()`** `std::optional< FormulaT > smtrat::CardinalityEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]`

Implements `smtrat::PseudoBoolEncoder`.

**0.15.79.2.3 `encode()`** `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

#### Returns

encoded formula

**0.15.79.2.4 encodingSize()** `Rational smtrat::CardinalityEncoder::encodingSize (`  
    `const ConstraintT & constraint ) [virtual]`  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

**0.15.79.2.5 generateVarChain()** `FormulaT smtrat::PseudoBoolEncoder::generateVarChain (`  
    `const std::set< carl::Variable > & vars,`  
    `carl::FormulaType type ) [protected], [inherited]`

**0.15.79.2.6 name()** `std::string smtrat::CardinalityEncoder::name ( ) [inline], [virtual]`  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

### 0.15.79.3 Field Documentation

**0.15.79.3.1 problem\_size** `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

## 0.15.80 smtrat::cadcells::representation::cell< H > Struct Template Reference

Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds.  
`#include <heuristics.h>`

### Static Public Member Functions

- `template<typename T >`  
`static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)`

### 0.15.80.1 Detailed Description

```
template<CellHeuristic H>
struct smtrat::cadcells::representation::cell< H >
```

Note: If connected(i) holds, then the indexed root ordering must contain an ordering between the interval bounds.

### 0.15.80.2 Member Function Documentation

**0.15.80.2.1 compute()** `template<CellHeuristic H>`  
`template<typename T >`  
`static std::optional< datastructures::CellRepresentation< T > > smtrat::cadcells::representation::cell< H >::compute (`  
`datastructures::SampledDerivationRef< T > & der ) [static]`

## 0.15.81 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST\_CELL > Struct Reference

```
#include <heuristics_cell.h>
```

### Static Public Member Functions

- `template<typename T >`  
`static std::optional< datastructures::CellRepresentation< T > > compute (datastructures::SampledDerivationRef< T > &der)`

### 0.15.81.1 Member Function Documentation

```
0.15.81.1.1 compute() template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::BIGGEST_CELL >::compute (
 datastructures::SampledDerivationRef< T > & der) [inline], [static]
```

## 0.15.82 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST\_CELL\_APPROXIMATION > Struct Reference

```
#include <heuristics_approximation.h>
```

### Static Public Member Functions

- template<typename T >
 static std::optional< datastructures::CellRepresentation< T > > **compute** (datastructures::SampledDerivationRef< T > &der)

### 0.15.82.1 Member Function Documentation

```
0.15.82.1.1 compute() template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::BIGGEST_CELL_APPROXIMATION >::compute (
 datastructures::SampledDerivationRef< T > & der) [inline], [static]
```

## 0.15.83 smtrat::cadcells::representation::cell< CellHeuristic::BIGGEST\_CELL\_EW > Struct Reference

```
#include <heuristics_cell.h>
```

### Static Public Member Functions

- template<typename T >
 static std::optional< datastructures::CellRepresentation< T > > **compute** (datastructures::SampledDerivationRef< T > &der)

### 0.15.83.1 Member Function Documentation

```
0.15.83.1.1 compute() template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::BIGGEST_CELL_EW >::compute (
 datastructures::SampledDerivationRef< T > & der) [inline], [static]
```

## 0.15.84 smtrat::cadcells::representation::cell< CellHeuristic::CHAIN\_EQ > Struct Reference

```
#include <heuristics_cell.h>
```

### Static Public Member Functions

- template<typename T >
 static std::optional< datastructures::CellRepresentation< T > > **compute** (datastructures::SampledDerivationRef< T > &der)

### 0.15.84.1 Member Function Documentation

```
0.15.84.1.1 compute() template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::CHAIN_EQ >::compute (
 datastructures::SampledDerivationRef< T > & der) [inline], [static]
```

## 0.15.85 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST\_DEGREE\_BARRIERS > Struct Reference

```
#include <heuristics_cell.h>
```

### Static Public Member Functions

- template<typename T >
 static std::optional< datastructures::CellRepresentation< T > > **compute** (datastructures::SampledDerivationRef< T > &der)

### 0.15.85.1 Member Function Documentation

```
0.15.85.1.1 compute() template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::LOWEST_DEGREE_BARRIERS >::compute (
 datastructures::SampledDerivationRef< T > & der) [inline], [static]
```

## 0.15.86 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST\_DEGREE\_BARRIERS\_EQ > Struct Reference

```
#include <heuristics_cell.h>
```

### Static Public Member Functions

- template<typename T >
 static std::optional< datastructures::CellRepresentation< T > > **compute** (datastructures::SampledDerivationRef< T > &der)

### 0.15.86.1 Member Function Documentation

```
0.15.86.1.1 compute() template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::LOWEST_DEGREE_BARRIERS_EQ >::compute (
 datastructures::SampledDerivationRef< T > & der) [inline], [static]
```

## 0.15.87 smtrat::cadcells::representation::cell< CellHeuristic::LOWEST\_DEGREE\_BARRIERS\_EW > Struct Reference

```
#include <heuristics_cell.h>
```

### Static Public Member Functions

- template<typename T >
 static std::optional< datastructures::CellRepresentation< T > > **compute** (datastructures::SampledDerivationRef< T > &der)

### 0.15.87.1 Member Function Documentation

```
0.15.87.1.1 compute() template<typename T >
static std::optional<datastructures::CellRepresentation<T> > smtrat::cadcells::representation::cell<
CellHeuristic::LOWEST_DEGREE_BARRIERS_EW >::compute (
 datastructures::SampledDerivationRef< T > & der) [inline], [static]
```

## 0.15.88 smtrat::cadcells::operators::properties::cell\_connected Struct Reference

```
#include <properties.h>
```

### Public Member Functions

- std::size\_t **level** () const
- std::size\_t **hash\_on\_level** () const

### Data Fields

- std::size\_t **lvl**

### Static Public Attributes

- static constexpr bool **is\_flag** = true

### 0.15.88.1 Member Function Documentation

```
0.15.88.1.1 hash_on_level() std::size_t smtrat::cadcells::operators::properties::cell_connected::hash_on_level () const [inline]
```

```
0.15.88.1.2 level() std::size_t smtrat::cadcells::operators::properties::cell_connected::level () const [inline]
```

### 0.15.88.2 Field Documentation

```
0.15.88.2.1 is_flag constexpr bool smtrat::cadcells::operators::properties::cell_connected::is_flag = true [static], [constexpr]
```

```
0.15.88.2.2 lvl std::size_t smtrat::cadcells::operators::properties::cell_connected::lvl
```

## 0.15.89 smtrat::cadcells::representation::approximation::CellApproximator Class Reference

```
#include <CellApproximator.h>
```

### Public Member Functions

- template<typename T > **CellApproximator** (datastructures::SampledDerivationRef< T > &der)
- **IR approximate\_bound** (const **IR** &p, const **RAN** &bound, bool below)
- **IR approximate\_between** (const **IR** &p\_l, const **IR** &p\_u, const **RAN** &l, const **RAN** &u)
- **datastructures::SymbolicInterval compute\_cell** ()

### 0.15.89.1 Constructor & Destructor Documentation

**0.15.89.1.1 CellApproximator()** template<typename T>  
smtrat::cadcells::representation::approximation::CellApproximator::CellApproximator(  
    datastructures::SampledDerivationRef< T > & der) [inline]

### 0.15.89.2 Member Function Documentation

**0.15.89.2.1 approximate\_between()** IR smtrat::cadcells::representation::approximation::CellApproximator::approximate\_between(  
    const IR & p\_l,  
    const IR & p\_u,  
    const RAN & l,  
    const RAN & u) [inline]

**0.15.89.2.2 approximate\_bound()** IR smtrat::cadcells::representation::approximation::CellApproximator::approximate\_bound(  
    const IR & p,  
    const RAN & bound,  
    bool below) [inline]

**0.15.89.2.3 compute\_cell()** datastructures::SymbolicInterval smtrat::cadcells::representation::approximation::CellApproximator::compute\_cell() [inline]

## 0.15.90 smtrat::cadcells::datastructures::CellRepresentation< P > Struct Template Reference

Represents a cell.

```
#include <representation.h>
```

### Public Member Functions

- [CellRepresentation \(SampledDerivationRef< P > &deriv\)](#)

### Data Fields

- [SymbolicInterval description](#)  
*Description of a cell.*
- [IndexedRootOrdering ordering](#)  
*An ordering on the roots that protects the cell.*
- [boost::container::flat\\_set< PolyRef > equational](#)  
*Polynomials that should be projected using the equational constraints projection.*
- [SampledDerivationRef< P > derivation](#)  
*Derivation.*

### 0.15.90.1 Detailed Description

```
template<typename P>
struct smtrat::cadcells::datastructures::CellRepresentation< P >
```

Represents a cell.

### 0.15.90.2 Constructor & Destructor Documentation

**0.15.90.2.1 CellRepresentation()** template<typename P >  
`smtrat::cadcells::datastructures::CellRepresentation< P >::CellRepresentation (`  
`SampledDerivationRef< P > & deriv ) [inline]`

### 0.15.90.3 Field Documentation

**0.15.90.3.1 derivation** template<typename P >  
`SampledDerivationRef<P> smtrat::cadcells::datastructures::CellRepresentation< P >::derivation`  
 Derivation.

**0.15.90.3.2 description** template<typename P >  
`SymbolicInterval smtrat::cadcells::datastructures::CellRepresentation< P >::description`  
 Description of a cell.

**0.15.90.3.3 equational** template<typename P >  
`boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures::CellRepresentation< P >::equational`  
 Polynomials that should be projected using the equational constraints projection.

**0.15.90.3.4 ordering** template<typename P >  
`IndexedRootOrdering smtrat::cadcells::datastructures::CellRepresentation< P >::ordering`  
 An ordering on the roots that protects the cell.

## 0.15.91 delta::Checker Class Reference

This class takes care of calling the actual solver.

```
#include <Checker.h>
```

### Public Member Functions

- **Checker** (const std::string &exec, std::size\_t memout, std::size\_t timeout, const std::string &original)  
*Creates a new checker.*
- bool **operator()** (const **Node** &n, const std::string &temp) const  
*Call the solver with the given node.*
- int **expectedCode** () const  
*Return expected exit code.*
- std::size\_t **getKilled** () const
- void **resetKilled** () const
- void **resetTimeout** (std::size\_t t)

### 0.15.91.1 Detailed Description

This class takes care of calling the actual solver.

### 0.15.91.2 Constructor & Destructor Documentation

```
0.15.91.2.1 Checker() delta::Checker::Checker (
 const std::string & exec,
 std::size_t memout,
 std::size_t timeout,
 const std::string & original) [inline]
```

Creates a new checker.

#### Parameters

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| <i>exec</i>     | Filename of the solver executable.                              |
| <i>timeout</i>  | Timeout for a single call to the solver.                        |
| <i>original</i> | Filename of the original file to obtain the expected exit code. |

### 0.15.91.3 Member Function Documentation

```
0.15.91.3.1 expectedCode() int delta::Checker::expectedCode () const [inline]
```

Return expected exit code.

#### Returns

Expected exit code.

```
0.15.91.3.2 getKilled() std::size_t delta::Checker::getKilled () const [inline]
```

```
0.15.91.3.3 operator() bool delta::Checker::operator() (
 const Node & n,
 const std::string & temp) const [inline]
```

Call the solver with the given node.

Checks if the exit code is the same as for the original file.

#### Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>n</i>    | Input data.                     |
| <i>temp</i> | Filename of the temporary file. |

#### Returns

If the exit code is still the same.

```
0.15.91.3.4 resetKilled() void delta::Checker::resetKilled () const [inline]
```

```
0.15.91.3.5 resetTimeout() void delta::Checker::resetTimeout (
 std::size_t t) [inline]
```

## 0.15.92 **benchmax::slurm::ChunkedSubmitfileProperties Struct Reference**

All properties needed to create a submit file.

```
#include <SlurmUtilities.h>
```

## Data Fields

- std::string **file\_suffix**  
*Suffix for job and submit file.*
- std::string **filename\_jobs**  
*Filename of the job list file.*
- std::string **tmp\_dir**  
*Temporary directory for log files.*
- carl::settings::duration **limit\_time**  
*Time limit in seconds.*
- carl::settings::duration **grace\_time**  
*Grace time in seconds.*
- carl::settings::binary\_quantity **limit\_memory**  
*Memory limit in megabytes.*
- std::size\_t **array\_size**  
*Array size.*
- std::size\_t **slice\_size**  
*Slice size.*
- std::pair< std::size\_t, std::size\_t > **job\_range**  
*This slice size.*

### 0.15.92.1 Detailed Description

All properties needed to create a submit file.

### 0.15.92.2 Field Documentation

**0.15.92.2.1 array\_size** std::size\_t benchmax::slurm::ChunkedSubmitfileProperties::array\_size  
Array size.

**0.15.92.2.2 file\_suffix** std::string benchmax::slurm::ChunkedSubmitfileProperties::file\_suffix  
Suffix for job and submit file.

**0.15.92.2.3 filename\_jobs** std::string benchmax::slurm::ChunkedSubmitfileProperties::filename\_jobs  
Filename of the job list file.

**0.15.92.2.4 grace\_time** carl::settings::duration benchmax::slurm::ChunkedSubmitfileProperties::grace\_time  
Grace time in seconds.

**0.15.92.2.5 job\_range** std::pair<std::size\_t, std::size\_t> benchmax::slurm::ChunkedSubmitfileProperties::job\_range  
This slice size.

**0.15.92.2.6 limit\_memory** carl::settings::binary\_quantity benchmax::slurm::ChunkedSubmitfileProperties::limit\_memory  
Memory limit in megabytes.

**0.15.92.2.7 limit\_time** carl::settings::duration benchmax::slurm::ChunkedSubmitfileProperties::limit\_time  
Time limit in seconds.

**0.15.92.2.8 slice\_size** std::size\_t benchmax::slurm::ChunkedSubmitfileProperties::slice\_size  
Slice size.

**0.15.92.2.9 tmp\_dir** std::string benchmax::slurm::ChunkedSubmitfileProperties::tmp\_dir  
Temporary directory for log files.

## 0.15.93 Minisat::Clause Class Reference

```
#include <SolverTypes.h>
```

### Public Member Functions

- void [calcAbstraction\(\)](#)
- int [size\(\)](#) const
- void [shrink\(int i\)](#)
- void [pop\(\)](#)
- unsigned [type\(\)](#) const
- bool [learnt\(\)](#) const
- bool [has\\_extra\(\)](#) const
- uint32\_t [mark\(\)](#) const
- void [mark\(uint32\\_t m\)](#)
- const [Lit & last\(\)](#) const
- bool [relocated\(\)](#) const
- [CRef relocation\(\)](#) const
- void [relocate\(CRef c\)](#)
- [Lit & operator\[\]\(int i\)](#)
- [Lit operator\[\]\(int i\)](#) const
- [operator const Lit \\* \(void\)](#) const
- float & [activity\(\)](#)
- float [activity\(\)](#) const
- uint32\_t [abstraction\(\)](#) const
- [Lit subsumes\(const Clause &other\)](#) const
- void [strengthen\(Lit p\)](#)

### Friends

- class [ClauseAllocator](#)
- template<class V>  
[Clause \\* Clause\\_new\(const V &p, bool learnt=false\)](#)

### 0.15.93.1 Member Function Documentation

**0.15.93.1.1 abstraction()** uint32\_t Minisat::Clause::abstraction() const [inline]

**0.15.93.1.2 activity() [1/2]** float& Minisat::Clause::activity() [inline]

**0.15.93.1.3 `activity()` [2/2]** float Minisat::Clause::activity ( ) const [inline]

**0.15.93.1.4 `calcAbstraction()`** void Minisat::Clause::calcAbstraction ( ) [inline]

**0.15.93.1.5 `has_extra()`** bool Minisat::Clause::has\_extra ( ) const [inline]

**0.15.93.1.6 `last()`** const `Lit&` Minisat::Clause::last ( ) const [inline]

**0.15.93.1.7 `learnt()`** bool Minisat::Clause::learnt ( ) const [inline]

**0.15.93.1.8 `mark()` [1/2]** uint32\_t Minisat::Clause::mark ( ) const [inline]

**0.15.93.1.9 `mark()` [2/2]** void Minisat::Clause::mark ( uint32\_t *m* ) [inline]

**0.15.93.1.10 `operator const Lit *()`** Minisat::Clause::operator const `Lit` \* ( void ) const [inline]

**0.15.93.1.11 `operator[]()` [1/2]** `Lit&` Minisat::Clause::operator[] ( int *i* ) [inline]

**0.15.93.1.12 `operator[]()` [2/2]** `Lit` Minisat::Clause::operator[] ( int *i* ) const [inline]

**0.15.93.1.13 `pop()`** void Minisat::Clause::pop ( ) [inline]

**0.15.93.1.14 `relocate()`** void Minisat::Clause::relocate ( `CRef` *c* ) [inline]

**0.15.93.1.15 `relocation()`** `CRef` Minisat::Clause::relocation ( ) const [inline]

**0.15.93.1.16 `reloced()`** bool Minisat::Clause::reloced ( ) const [inline]

**0.15.93.1.17 `shrink()`** void Minisat::Clause::shrink ( int *i* ) [inline]

**0.15.93.1.18 `size()`** int Minisat::Clause::size ( ) const [inline]

**0.15.93.1.19 `strengthen()`** void Minisat::Clause::strengthen (   
    Lit p ) [inline]

**0.15.93.1.20 `subsumes()`** Lit Minisat::Clause::subsumes (   
    const Clause & other ) const [inline]

**0.15.93.1.21 `type()`** unsigned Minisat::Clause::type ( ) const [inline]

## 0.15.93.2 Friends And Related Function Documentation

**0.15.93.2.1 `Clause_new`** template<class V>  
Clause\* Clause\_new (   
    const V & ps,   
    bool learnt = false ) [friend]

**0.15.93.2.2 `ClauseAllocator`** friend class ClauseAllocator [friend]

## 0.15.93.3 Field Documentation

**0.15.93.3.1 `abs`** uint32\_t Minisat::Clause::abs

**0.15.93.3.2 `act`** float Minisat::Clause::act

**0.15.93.3.3 `has_extra`** unsigned Minisat::Clause::has\_extra

**0.15.93.3.4 `lit`** Lit Minisat::Clause::lit

**0.15.93.3.5 `mark`** unsigned Minisat::Clause::mark

**0.15.93.3.6 `rel`** CRef Minisat::Clause::rel

**0.15.93.3.7 `relocoed`** unsigned Minisat::Clause::relocoed

**0.15.93.3.8 `size`** unsigned Minisat::Clause::size

**0.15.93.3.9 `type`** unsigned Minisat::Clause::type

## 0.15.94 Minisat::ClauseAllocator Class Reference

#include <SolverTypes.h>

## Public Types

- enum
- enum
- typedef uint32\_t Ref

## Public Member Functions

- ClauseAllocator (uint32\_t start\_cap)
- ClauseAllocator ()
- void moveTo (ClauseAllocator &to)
- template<class Lits >  
  CRef alloc (const Lits &ps, unsigned \_type)
- Clause & operator[] (Ref r)
- const Clause & operator[] (Ref r) const
- Clause \* lea (Ref r)
- const Clause \* lea (Ref r) const
- Ref ael (const Clause \*t)
- void free (CRef cid)
- void reloc (CRef &cr, ClauseAllocator &to)
- uint32\_t size () const
- uint32\_t wasted () const
- Ref alloc (int size)
- void free (int size)
- Ref ael (const uint32\_t \*t)
- void moveTo (RegionAllocator &to)

## Data Fields

- bool extra\_clause\_field

### 0.15.94.1 Member Typedef Documentation

**0.15.94.1.1 Ref** typedef uint32\_t Minisat::RegionAllocator< uint32\_t >::Ref [inherited]

### 0.15.94.2 Member Enumeration Documentation

**0.15.94.2.1 anonymous enum** anonymous enum [inherited]

**0.15.94.2.2 anonymous enum** anonymous enum [inherited]

### 0.15.94.3 Constructor & Destructor Documentation

**0.15.94.3.1 ClauseAllocator() [1/2]** Minisat::ClauseAllocator::ClauseAllocator (  
  uint32\_t start\_cap ) [inline]

**0.15.94.3.2 ClauseAllocator() [2/2]** Minisat::ClauseAllocator::ClauseAllocator ( ) [inline]

#### 0.15.94.4 Member Function Documentation

**0.15.94.4.1 `ael()` [1/2]** `Ref Minisat::ClauseAllocator::ael (`  
    `const Clause * t )` [inline]

**0.15.94.4.2 `ael()` [2/2]** `Ref Minisat::RegionAllocator< uint32_t >::ael (`  
    `const uint32_t * t )` [inherited]

**0.15.94.4.3 `alloc()` [1/2]** `template<class Lits >`  
`CRef Minisat::ClauseAllocator::alloc (`  
    `const Lits & ps,`  
    `unsigned _type )` [inline]

**0.15.94.4.4 `alloc()` [2/2]** `RegionAllocator< uint32_t >::Ref Minisat::RegionAllocator< uint32_t >::alloc (`  
    `int size )` [inherited]

**0.15.94.4.5 `free()` [1/2]** `void Minisat::ClauseAllocator::free (`  
    `CRef cid )` [inline]

**0.15.94.4.6 `free()` [2/2]** `void Minisat::RegionAllocator< uint32_t >::free (`  
    `int size )` [inherited]

**0.15.94.4.7 `lea()` [1/2]** `Clause* Minisat::ClauseAllocator::lea (`  
    `Ref r )` [inline]

**0.15.94.4.8 `lea()` [2/2]** `const Clause* Minisat::ClauseAllocator::lea (`  
    `Ref r ) const` [inline]

**0.15.94.4.9 `moveTo()` [1/2]** `void Minisat::ClauseAllocator::moveTo (`  
    `ClauseAllocator & to )` [inline]

**0.15.94.4.10 `moveTo()` [2/2]** `void Minisat::RegionAllocator< uint32_t >::moveTo (`  
    `RegionAllocator< uint32_t > & to )` [inherited]

**0.15.94.4.11 `operator[]()` [1/2]** `Clause& Minisat::ClauseAllocator::operator[] (`  
    `Ref r )` [inline]

**0.15.94.4.12 `operator[]()` [2/2]** `const Clause& Minisat::ClauseAllocator::operator[] (`  
    `Ref r ) const` [inline]

**0.15.94.4.13 `reloc()`** void Minisat::ClauseAllocator::reloc ( CRef & cr, ClauseAllocator & to ) [inline]

**0.15.94.4.14 `size()`** uint32\_t Minisat::RegionAllocator< uint32\_t >::size ( void ) const [inline], [inherited]

**0.15.94.4.15 `wasted()`** uint32\_t Minisat::RegionAllocator< uint32\_t >::wasted ( ) const [inline], [inherited]

## 0.15.94.5 Field Documentation

**0.15.94.5.1 `extra_clause_field`** bool Minisat::ClauseAllocator::extra\_clause\_field

## 0.15.95 smrat::mcsat::ClauseChain Class Reference

An explanation is either a single clause or a chain of clauses, satisfying the following properties:

```
#include <ClauseChain.h>
```

### Data Structures

- struct [Link](#)

### Public Types

- using `const_iterator` = typename std::vector<[Link](#)>::const\_iterator

### Public Member Functions

- `ClauseChain ()`
- `FormulaT createTseitinVar (const FormulaT &formula)`

*Create a Tseitin variable for the given formula.*
- `void appendPropagating (const FormulaT &&clause, const FormulaT &&impliedTseitinLiteral)`

*Append a clause that implies impliedTseitinLiteral under the current assignment.*
- `void appendConflicting (const FormulaT &&clause)`

*Append a conflicting clause (regarding the current assignment).*
- `void appendOptional (const FormulaT &&clause)`

*Append an additional clause which is neither propagating nor conflicting.*
- `const std::vector<Link> & chain () const`
- `const_iterator begin () const`
- `const_iterator end () const`
- `FormulaT resolve () const`

*Performs resolution on the chain.*
- `FormulaT to_formula () const`

*Transforms the clause chain into a formula (containing Tseitin variables).*

### Static Public Member Functions

- static `ClauseChain from_formula (const FormulaT &f, const Model &model, bool with_equivalence)`

*Transforms a given Boolean conjunction over AND and OR to CNF via Tseitin-Transformation so that, if the input formula is conflicting under the current assignment, after all clauses in "implications" have been propagated in the given order, the returned formula evaluates to false.*

## Friends

- std::ostream & `operator<<` (std::ostream &stream, const ClauseChain &chain)

### 0.15.95.1 Detailed Description

An explanation is either a single clause or a chain of clauses, satisfying the following properties:

- If the clauses are inserted and propagated in the chain's order, at least the last clause should be conflicting.
- Each conflicting clause is already conflicting after preceding clauses have been inserted and propagated.
- Tseitin vars should be used so that a Tseitin variable C is equivalent to its corresponding subformula F (or at least  $C \rightarrow F$ ).

### 0.15.95.2 Member Typedef Documentation

**0.15.95.2.1 const\_iterator** using `smtrat::mcsat::ClauseChain::const_iterator` = typename std::vector<[Link](#)>::const\_iterator

### 0.15.95.3 Constructor & Destructor Documentation

**0.15.95.3.1 ClauseChain()** `smtrat::mcsat::ClauseChain::ClauseChain()` [inline]

### 0.15.95.4 Member Function Documentation

**0.15.95.4.1 appendConflicting()** `void smtrat::mcsat::ClauseChain::appendConflicting(const FormulaT && clause)` [inline]  
Append a conflicting clause (regarding the current assignment).

**0.15.95.4.2 appendOptional()** `void smtrat::mcsat::ClauseChain::appendOptional(const FormulaT && clause)` [inline]  
Append an additional clause which is neither propagating nor conflicting.  
Can be used to pass additional knowledge.

**0.15.95.4.3 appendPropagating()** `void smtrat::mcsat::ClauseChain::appendPropagating(const FormulaT && clause, const FormulaT && impliedTseitinLiteral)` [inline]  
Append a clause that implies impliedTseitinLiteral under the current assignment.  
Note that impliedTseitinLiteral =  $\sim v$  for a Tseitin variable v obtained via createTseitinVar.

**0.15.95.4.4 begin()** `const_iterator smtrat::mcsat::ClauseChain::begin()` const [inline]  
Returns

A constant iterator to the beginning of this chain.

**0.15.95.4.5 chain()** `const std::vector<Link>& smtrat::mcsat::ClauseChain::chain()` const [inline]  
Returns

A vector representing this chain.

**0.15.95.4.6 `createTseitinVar()`** `FormulaT smtrat::mcsat::ClauseChain::createTseitinVar ( const FormulaT & formula ) [inline]`

Create a Tseitin variable for the given formula.

The returned variable C should be used so that  $C \leftrightarrow formula$  or at least  $\sim formula \rightarrow \sim C$

**0.15.95.4.7 `end()`** `const_iterator smtrat::mcsat::ClauseChain::end () const [inline]`

Returns

A constant iterator to the end of this chain.

**0.15.95.4.8 `from_formula()`** `ClauseChain smtrat::mcsat::ClauseChain::from_formula ( const FormulaT & f, const Model & model, bool with_equivalence ) [static]`

Transforms a given Boolean conjunction over AND and OR to CNF via Tseitin-Transformation so that, if the input formula is conflicting under the current assignment, after all clauses in "implications" have been propagated in the given order, the returned formula evaluates to false.

**0.15.95.4.9 `resolve()`** `FormulaT smtrat::mcsat::ClauseChain::resolve () const`

Performs resolution on the chain.

Uses the last clause for resolution.

Returns

A single clause conflicting under the current assignment.

**0.15.95.4.10 `to_formula()`** `FormulaT smtrat::mcsat::ClauseChain::to_formula () const`

Transforms the clause chain into a formula (containing Tseitin variables).

## 0.15.95.5 Friends And Related Function Documentation

**0.15.95.5.1 `operator<<`** `std::ostream& operator<< ( std::ostream & stream, const ClauseChain & chain ) [friend]`

## 0.15.96 smtrat::sat::detail::ClauseChecker Struct Reference

```
#include <ClauseChecker.h>
```

### Public Member Functions

- `Model buildModel () const`
- `void operator() (const FormulaT &formula) const`
- `void operator() (const FormulasT &formulas) const`
- `template<typename VM , typename BCM >`  
`void operator() (const Minisat::Clause &c, const VM &vm, const BCM &bcm) const`

### 0.15.96.1 Member Function Documentation

**0.15.96.1.1 buildModel()** `Model smtrat::sat::detail::ClauseChecker::buildModel () const [inline]`

**0.15.96.1.2 operator()() [1/3]** `void smtrat::sat::detail::ClauseChecker::operator() (const FormulasT & formulas) const [inline]`

**0.15.96.1.3 operator()() [2/3]** `void smtrat::sat::detail::ClauseChecker::operator() (const FormulaT & formula) const [inline]`

**0.15.96.1.4 operator()() [3/3]** `template<typename VM, typename BCM> void smtrat::sat::detail::ClauseChecker::operator() (const Minisat::Clause & c, const VM & vm, const BCM & bcm) const [inline]`

## 0.15.97 smtrat::CMakeOptionPrinter Struct Reference

#include <compile\_information.h>

### Data Fields

- bool `advanced`

#### 0.15.97.1 Field Documentation

**0.15.97.1.1 advanced** `bool smtrat::CMakeOptionPrinter::advanced`

## 0.15.98 Minisat::CMap< T > Class Template Reference

#include <SolverTypes.h>

### Public Member Functions

- void `clear ()`
- int `size () const`
- void `insert (CRef cr, const T &t)`
- void `growTo (CRef cr, const T &t)`
- void `remove (CRef cr)`
- bool `has (CRef cr, T &t)`
- const T & `operator[] (CRef cr) const`
- T & `operator[] (CRef cr)`
- int `bucket_count () const`
- const `vec< typename HashTable::Pair > & bucket (int i) const`
- void `moveTo (CMap &other)`
- void `debug ()`

#### 0.15.98.1 Member Function Documentation

**0.15.98.1.1 bucket()** `template<class T> const vec<typename HashTable::Pair>& Minisat::CMap< T >::bucket (int i) const [inline]`

**0.15.98.1.2 `bucket_count()`** template<class T >  
int Minisat::CMap< T >::bucket\_count ( ) const [inline]

**0.15.98.1.3 `clear()`** template<class T >  
void Minisat::CMap< T >::clear ( ) [inline]

**0.15.98.1.4 `debug()`** template<class T >  
void Minisat::CMap< T >::debug ( ) [inline]

**0.15.98.1.5 `growTo()`** template<class T >  
void Minisat::CMap< T >::growTo (   
    CRef cr,  
    const T & t ) [inline]

**0.15.98.1.6 `has()`** template<class T >  
bool Minisat::CMap< T >::has (   
    CRef cr,  
    T & t ) [inline]

**0.15.98.1.7 `insert()`** template<class T >  
void Minisat::CMap< T >::insert (   
    CRef cr,  
    const T & t ) [inline]

**0.15.98.1.8 `moveTo()`** template<class T >  
void Minisat::CMap< T >::moveTo (   
    CMap< T > & other ) [inline]

**0.15.98.1.9 `operator[]()` [1/2]** template<class T >  
T& Minisat::CMap< T >::operator[] (   
    CRef cr ) [inline]

**0.15.98.1.10 `operator[]()` [2/2]** template<class T >  
const T& Minisat::CMap< T >::operator[] (   
    CRef cr ) const [inline]

**0.15.98.1.11 `remove()`** template<class T >  
void Minisat::CMap< T >::remove (   
    CRef cr ) [inline]

**0.15.98.1.12 `size()`** template<class T >  
int Minisat::CMap< T >::size ( ) const [inline]

## 0.15.99 smrat::CNFerModule Class Reference

```
#include <CNFerModule.h>
```

## Data Structures

- struct [SettingsType](#)

## Public Types

- enum class [LemmaType](#) : unsigned { [NORMAL](#) = 0 , [PERMANENT](#) = 1 }

## Public Member Functions

- [CNFerModule](#) (const [ModuleInput](#) \*, [Conditionals](#) &, [Manager](#) \*const =NULL)  
*Constructs a [CNFerModule](#).*
- [~CNFerModule](#) ()  
*Destructs a [CNFerModule](#).*
- [Answer checkCore](#) ()  
*Checks the received formula for consistency.*
- bool [isPreprocessor](#) () const
- bool [appliedPreprocessing](#) () const
- bool [add](#) ([ModuleInput](#)::const\_iterator \_subformula)  
*The module has to take the given sub-formula of the received formula into account.*
- void [remove](#) ([ModuleInput](#)::const\_iterator \_subformula)  
*Removes everything related to the given sub-formula of the received formula.*
- [Answer check](#) (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)  
*Checks the received formula for consistency.*
- [Answer runBackends](#) (bool \_final, bool \_full, carl::Variable \_objective)  
*Runs the backend solvers on the passed formula.*
- virtual [Answer runBackends](#) ()
- std::pair< bool, [FormulaT](#) > [getReceivedFormulaSimplified](#) ()
- void [updateModel](#) () const  
*Updates the current assignment into the model.*
- bool [inform](#) (const [FormulaT](#) &\_constraint)  
*Informs the module about the given constraint.*
- void [deinform](#) (const [FormulaT](#) &\_constraint)  
*The inverse of informing about a constraint.*
- virtual void [init](#) ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- virtual void [updateAllModels](#) ()  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > [getModelEqualities](#) () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned [currentlySatisfiedByBackend](#) (const [FormulaT](#) &\_formula) const
- virtual unsigned [currentlySatisfied](#) (const [FormulaT](#) &) const
- bool [receivedVariable](#) (carl::Variable::Arg \_var) const
- [Answer solverState](#) () const
- std::size\_t [id](#) () const
- void [setId](#) (std::size\_t \_id)  
*Sets this modules unique ID to identify itself.*
- [thread\\_priority threadPriority](#) () const
- void [setThreadPriority](#) ([thread\\_priority](#) \_threadPriority)  
*Sets the priority of this module to get a thread for running its check procedure.*
- const [ModuleInput](#) \* [pReceivedFormula](#) () const
- const [ModuleInput](#) & [rReceivedFormula](#) () const
- const [ModuleInput](#) \* [pPassedFormula](#) () const

- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const `std::vector< Model > & allModels () const`
- const `std::vector< FormulaSetT > & infeasibleSubsets () const`
- const `std::vector< Module * > & usedBackends () const`
- const `carl::FastSet< FormulaT > & constraintsToInform () const`
- const `carl::FastSet< FormulaT > & informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const `std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- virtual `std::string moduleName () const`
- `carl::Variable objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `isValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const FormulaT &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- virtual bool `addCore` (ModuleInput::const\_iterator formula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (ModuleInput::const\_iterator formula)  
*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- ModuleInput::iterator `passedFormulaBegin` ()
- ModuleInput::iterator `passedFormulaEnd` ()
- void `addOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const FormulaT & `getOrigins` (ModuleInput::const\_iterator \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const FormulaT &\_formula, FormulasT &\_origins) const
- void `getOrigins` (const FormulaT &\_formula, FormulaSetT &\_origins) const
- std::pair< ModuleInput::iterator, bool > `removeOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*ptr< std::vector< FormulaT >> &\_origins)*
- std::pair< ModuleInput::iterator, bool > `removeOrigins` (ModuleInput::iterator \_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)
- void `informBackends` (const FormulaT &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const FormulaT &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< ModuleInput::iterator, bool > `addReceivedSubformulaToPassedFormula` (ModuleInput::const\_iterator \_subformula)  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const FormulaT &\_origin) const
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula)  
*Adds the given formula to the passed formula with no origin.*

- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void [getBackendsModel](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
- void [getBackendsAllModels](#) () const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- virtual [ModuleInput::iterator](#) [eraseSubformulaFromPassedFormula](#) ([ModuleInput::iterator](#) \_subformula, bool \_ignoreOrigins=false)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void [clearPassedFormula](#) ()
 

*Merges the two vectors of sets into the first one.*
- std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
   
bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
   
bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())*
- void [splitUnequalConstraint](#) (const [FormulaT](#) &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned [checkModel](#) () const
 

*Adds a formula to the InformationRelevantFormula.*
- void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)
 

*Gets all InformationRelevantFormulas.*
- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector<FormulaSetT> mInfeasibleSubsets`

*Stores the infeasible subsets.*

- `Manager *const mpManager`

*A reference to the manager.*

- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*

- `std::vector<Model> mAllModels`

*Stores all satisfying assignments.*

- `bool mModelComputed`

*True, if the model has already been computed.*

- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*

- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*

- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*

- `std::vector<TheoryPropagation> mTheoryPropagations`

- `std::atomic<Answer> mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- `Conditionals mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- `std::vector<Module*> mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- `std::vector<Module*> mAllBackends`

*The backends of this module which have been used.*

- `std::vector<Lemma> mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*

- `ModuleInput::iterator mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- `carl::FastSet<FormulaT> mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet<FormulaT> mInformedConstraints`

*Stores the position of the first constraint of which no backend has been informed about.*

- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- `unsigned mSmallerMusesCheckCounter`

*Counter used for the generation of the smt2 files to check for smaller muses.*

- `std::vector<std::size_t> mVariableCounters`

*Maps variables to the number of their occurrences.*

**0.15.99.1 Member Enumeration Documentation****0.15.99.1.1 LemmaType** enum `smtrat::Module::LemmaType` : unsigned [strong], [inherited]

## Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

## 0.15.99.2 Constructor &amp; Destructor Documentation

**0.15.99.2.1 CNFerModule()** smtrat::CNFerModule::CNFerModule (

```
const ModuleInput * ,
Conditionals & ,
Manager * const = NULL)
```

Constructs a [CNFerModule](#).

**0.15.99.2.2 ~CNFerModule()** smtrat::CNFerModule::~CNFerModule ( )

Destructs a [CNFerModule](#).

## 0.15.99.3 Member Function Documentation

**0.15.99.3.1 add()** bool smtrat::PModule::add (

```
ModuleInput::const_iterator _subformula) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

## Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

## Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.99.3.2 addConstraintToInform()** void smtrat::Module::addConstraintToInform (

```
const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

## Parameters

|             |                        |
|-------------|------------------------|
| _constraint | The constraint to add. |
|-------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.99.3.3 addCore()** virtual bool smtrat::Module::addCore (

```
ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

The module has to take the given sub-formula of the received formula into account.

## Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| formula | The sub-formula to take additionally into account. |
|---------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.99.3.4 addInformationRelevantFormula()** void smrat::Module::addInformationRelevantFormula (

const [FormulaT](#) & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.99.3.5 addLemma()** void smrat::Module::addLemma (

const [FormulaT](#) & \_lemma,

const [LemmaType](#) & \_lt = [LemmaType::NORMAL](#),

const [FormulaT](#) & \_preferredFormula = [FormulaT\( carl::FormulaType::TRUE \)](#) ) [inline],

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.99.3.6 addOrigin()** void smrat::Module::addOrigin (

[ModuleInput::iterator](#) \_formula,

const [FormulaT](#) & \_origin ) [inline], [protected], [inherited]

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

**0.15.99.3.7 addReceivedSubformulaToPassedFormula()** std::pair<[ModuleInput::iterator](#),bool> smrat::Module::addReceivedSubformulaToPassedFormula (

---

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.99.3.8 addSubformulaToPassedFormula()** [1/3] `std::pair<ModuleInput::iterator,bool> smtrat→`  
`::Module::addSubformulaToPassedFormula (`

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.99.3.9 addSubformulaToPassedFormula()** [2/3] `std::pair<ModuleInput::iterator,bool> smtrat→`  
`::Module::addSubformulaToPassedFormula (`

```
 const FormulaT & _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.99.3.10 addSubformulaToPassedFormula()** [3/3] `std::pair<ModuleInput::iterator,bool> smtrat→`  
`::Module::addSubformulaToPassedFormula (`

```
 const FormulaT & _formula,
```

```
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
```

[inherited]

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.99.3.11 `allModels()`** `const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]`

**Returns**

All satisfying assignments, if existent.

**0.15.99.3.12 `anAnswerFound()`** `bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.99.3.13 `answerFound()`** `const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]`

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.99.3.14 `appliedPreprocessing()`** `bool smtrat::PModule::appliedPreprocessing () const [inline], [inherited]`

**Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.99.3.15 `backendsModel()`** `const Model & smtrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.99.3.16 branchAt() [1/4]** bool smrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.99.3.17 branchAt() [2/4]** bool smrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.99.3.18 branchAt() [3/4]** bool smrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.99.3.19 branchAt() [4/4]** bool smrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.99.3.20 check() Answer smrat::PModule::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

```
0.15.99.3.21 checkCore() Answer smrat::CNFerModule::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

```
0.15.99.3.22 checkInfSubsetForMinimality() void smrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.99.3.23 checkModel() unsigned smrat::Module::checkModel () const [protected], [inherited]
```

#### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.99.3.24 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.99.3.25 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.99.3.26 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.99.3.27 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.99.3.28 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.99.3.29 collectOrigins() [1/2]** void smtrat::Module::collectOrigins (

```
const FormulaT & _formula,
FormulaSetT & _origins) const [inherited]
```

**0.15.99.3.30 collectOrigins() [2/2]** void smtrat::Module::collectOrigins (

```
const FormulaT & _formula,
FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.99.3.31 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( )

[inherited]

**0.15.99.3.32 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

#### Returns

The constraints which the backends must be informed about.

```
0.15.99.3.33 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

```
0.15.99.3.34 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.99.3.35 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

```
0.15.99.3.36 deinformCore() virtual void smtrat::Module::deinformCore (
 const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

```
0.15.99.3.37 determine_smallest_origin() size_t smtrat::Module::determine_smallest_origin (
 const std::vector< FormulaT > & origins) const [protected], [inherited]
```

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

---

**0.15.99.3.38 eraseSubformulaFromPassedFormula()** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

`ModuleInput::iterator _subformula,`

`bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.99.3.39 excludeNotReceivedVariablesFromModel()** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.99.3.40 findBestOrigin()** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`

`const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

#### Returns

**0.15.99.3.41 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

#### Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.99.3.42 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

#### Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.99.3.43 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable (`

`const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.99.3.44 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
 Stores the trivial infeasible subset being the set of received formulas.

**0.15.99.3.45 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
 Stores all models of a backend in the list of all models of this module.

**0.15.99.3.46 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.99.3.47 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
 Copies the infeasible subsets of the passed formula.

**0.15.99.3.48 getInfeasibleSubsets() [2/2]** std::vector< [FormulaSetT](#) > smtrat::Module::getInfeasibleSubsets ( ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.99.3.49 getInformationRelevantFormulas()** const std::set< [FormulaT](#) > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
 Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.99.3.50 getModelEqualities()** std::list< std::vector< [carl::Variable](#) > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]  
 Partition the variables from the current model into equivalence classes according to their assigned value.  
 The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.99.3.51 getOrigins() [1/3]** void smtrat::Module::getOrigins ( ) const [inline], [protected], [inherited]

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.99.3.52 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins (`  
    `const FormulaT & _formula,`  
    `FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.99.3.53 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins (`  
    `ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.99.3.54 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.99.3.55 `hasLemmas()`** `bool smtrat::Module::hasLemmas ( ) [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.99.3.56 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.99.3.57 id()** std::size\_t smrat::Module::id ( ) const [inline], [inherited]

#### Returns

A unique ID to identify this module instance.

**0.15.99.3.58 infeasibleSubsets()** const std::vector<FormulaSetT>& smrat::Module::infeasibleSubsets ( ) const [inline], [inherited]

#### Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.99.3.59 inform()** bool smrat::Module::inform ( const FormulaT & \_constraint ) [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|             |                                 |
|-------------|---------------------------------|
| _constraint | The constraint to inform about. |
|-------------|---------------------------------|

#### Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.99.3.60 informBackends()** void smrat::Module::informBackends ( const FormulaT & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

#### Parameters

|             |                                 |
|-------------|---------------------------------|
| _constraint | The constraint to inform about. |
|-------------|---------------------------------|

**0.15.99.3.61 informCore()** virtual bool smrat::Module::informCore ( const FormulaT & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|             |                                 |
|-------------|---------------------------------|
| _constraint | The constraint to inform about. |
|-------------|---------------------------------|

#### Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.99.3.62 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.99.3.63 init()** `void smrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.99.3.64 is\_minimizing()** `bool smrat::Module::is_minimizing ( ) const [inline], [inherited]`

**0.15.99.3.65 isLemmaLevel()** `bool smrat::Module::isLemmaLevel ( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.99.3.66 isPreprocessor()** `bool smrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.99.3.67 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas ( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.99.3.68 merge()** `std::vector< FormulaT > smrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.99.3.69 `model()`** `const Model& smtrat::Module::model() const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.99.3.70 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint()`

```
const Model & _modelA,
const Model & _modelB) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.99.3.71 `moduleName()`** `virtual std::string smtrat::Module::moduleName() const [inline], [virtual], [inherited]`**Returns**

The name of the given module type as name.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SplitSOSModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LVEModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::GBModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQPreprocessingModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::BEModule< Settings >](#).

**0.15.99.3.72 `objective()`** `carl::Variable smtrat::Module::objective() const [inline], [inherited]`**0.15.99.3.73 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.99.3.74 passedFormulaBegin()** `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.99.3.75 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.99.3.76 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.99.3.77 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.99.3.78 print()** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const` [inherited]

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.99.3.79 printAllModels()** `void smtrat::Module::printAllModels (`  
`std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.99.3.80 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets (`  
`const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.99.3.81 `printModel()`** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.99.3.82 `printPassedFormula()`** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.99.3.83 `printReceivedFormula()`** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.99.3.84 `probablyLooping()`** bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & *\_branchingPolynomial*, const Rational & *\_branchingValue* ) const [protected], [inherited]

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

**Parameters**

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

**Returns**

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.99.3.85 `receivedFormulaChecked()`** void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.99.3.86 `receivedFormulasAsInfeasibleSubset()`** void smrat::Module::receivedFormulasAsInfeasibleSubset (

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

**0.15.99.3.87 `receivedVariable()`** bool smrat::Module::receivedVariable (

```
 carl::Variable::Arg _var) const [inline], [inherited]
```

**0.15.99.3.88 `remove()`** void smrat::PModule::remove (

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.99.3.89 `removeCore()`** virtual void smrat::Module::removeCore (

```
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.99.3.90 `removeOrigin()`** std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin (

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.99.3.91 `removeOrigins()`** std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins (

```
ModuleInput::iterator _formula,
const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.99.3.92 rPassedFormula()** const ModuleInput& smtrat::Module::rPassedFormula () const  
[inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.99.3.93 rReceivedFormula()** const ModuleInput& smtrat::Module::rReceivedFormula () const  
[inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.99.3.94 runBackends() [1/2]** virtual Answer smtrat::PModule::runBackends () [inline],  
[virtual], [inherited]

Reimplemented from [smtrat::Module](#).

**0.15.99.3.95 runBackends() [2/2]** Answer smtrat::PModule::runBackends (
 bool \_final,
 bool \_full,
 carl::Variable \_objective ) [virtual], [inherited]

Runs the backend solvers on the passed formula.

**Parameters**

|            |                                                                                                                                       |
|------------|---------------------------------------------------------------------------------------------------------------------------------------|
| _final     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| _full      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| _objective | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.99.3.96 setId()** void smtrat::Module::setId (
 std::size\_t \_id ) [inline], [inherited]

Sets this modules unique ID to identify itself.

**Parameters**

|                     |                                    |
|---------------------|------------------------------------|
| ↔<br>↔<br><i>id</i> | The id to set this module's id to. |
|---------------------|------------------------------------|

**0.15.99.3.97 `setThreadPriority()`** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`  
Sets the priority of this module to get a thread for running its check procedure.

Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.99.3.98 `solverState()`** `Answer smtrat::Module::solverState () const [inline], [inherited]`

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.99.3.99 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.99.3.100 `threadPriority()`** `thread_priority smtrat::Module::threadPriority () const [inline], [inherited]`

Returns

The priority of this module to get a thread for running its check procedure.

**0.15.99.3.101 `updateAllModels()`** `void smtrat::Module::updateAllModels () [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.99.3.102 `updateLemmas()`** `void smtrat::Module::updateLemmas () [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.99.3.103 `updateModel()`** `void smtrat::PModule::updateModel () const [virtual], [inherited]`  
Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.99.3.104 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends () const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.99.4 Field Documentation**

**0.15.99.4.1 mAllBackends** `std::vector<Module*> smrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.99.4.2 mAllModels** `std::vector<Model> smrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

**0.15.99.4.3 mBackendsFoundAnswer** `std::atomic_bool* smrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.99.4.4 mConstraintsToInform** `carl::FastSet<FormulaT> smrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

**0.15.99.4.5 mFinalCheck** `bool smrat::Module::mFinalCheck [protected], [inherited]`  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.99.4.6 mFirstPosInLastBranches** `std::size_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

**0.15.99.4.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.99.4.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.99.4.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.99.4.10 mFullCheck** `bool smrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.99.4.11 mInfeasibleSubsets** std::vector<[FormulaSetT](#)> smtrat::Module::mInfeasibleSubsets  
[protected], [inherited]  
Stores the infeasible subsets.

**0.15.99.4.12 mInformedConstraints** carl::FastSet<[FormulaT](#)> smtrat::Module::mInformedConstraints  
[protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.99.4.13 mLastBranches** std::vector< [Branching](#) > smtrat::Module::mLastBranches = std::vector<[Branching](#)> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static],  
[inherited]  
Stores the last branches in a cycle buffer.

**0.15.99.4.14 mLemmas** std::vector<[Lemma](#)> smtrat::Module::mLemmas [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.99.4.15 mModel** [Model](#) smtrat::Module::mModel [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.99.4.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected],  
[inherited]  
True, if the model has already been computed.

**0.15.99.4.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5  
[static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.99.4.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected],  
[inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.99.4.19 mOldSplittingVariables** std::vector< [FormulaT](#) > smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.99.4.20 mpManager** [Manager](#)\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.99.4.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.99.4.22 mSolverState** std::atomic<[Answer](#)> smtrat::Module::mSolverState [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

---

**0.15.99.4.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheory←  
Propagations [protected], [inherited]

**0.15.99.4.24 mUsedBackends** std::vector<Module\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.99.4.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.100 smtrat::CoCoAGBModule< Settings > Class Template Reference

```
#include <CoCoAGBModule.h>
```

### Public Types

- enum class **LemmaType** : unsigned { **NORMAL** = 0 , **PERMANENT** = 1 }

### Public Member Functions

- bool **inform** (const FormulaT &\_constraint)  
*Informs the module about the given constraint.*
- void **deinform** (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- virtual void **init** ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- bool **add** (ModuleInput::const\_iterator \_subformula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual **Answer check** (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO←  
VARIABLE)  
*Checks the received formula for consistency.*
- virtual void **remove** (ModuleInput::const\_iterator \_subformula)  
*Removes everything related to the given sub-formula of the received formula.*
- virtual void **updateModel** () const  
*Updates the model, if the solver has detected the consistency of the received formula, beforehand.*
- virtual void **updateAllModels** ()  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned **currentlySatisfiedByBackend** (const FormulaT &\_formula) const
- virtual unsigned **currentlySatisfied** (const FormulaT &) const
- bool **receivedVariable** (carl::Variable::Arg \_var) const
- Answer solverState** () const
- std::size\_t **id** () const
- void **setId** (std::size\_t \_id)  
*Sets this modules unique ID to identify itself.*
- thread\_priority threadPriority** () const
- void **setThreadPriority** (**thread\_priority** \_threadPriority)  
*Sets the priority of this module to get a thread for running its check procedure.*
- const ModuleInput \* **pReceivedFormula** () const

- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const `std::vector< Model > & allModels () const`
- const `std::vector< FormulaSetT > & infeasibleSubsets () const`
- const `std::vector< Module * > & usedBackends () const`
- const `carl::FastSet< FormulaT > & constraintsToInform () const`
- const `carl::FastSet< FormulaT > & informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const `std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- virtual `std::string moduleName () const`
- `carl::Variable objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `hasValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _insubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- void `print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` & \_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< `Branching` > `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< `FormulaT` > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` & \_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` & \_constraint)  
*The inverse of informing about a constraint.*
- virtual bool `addCore` (`ModuleInput::const_iterator` formula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual `Answer checkCore` ()  
*Checks the received formula for consistency.*
- virtual void `removeCore` (`ModuleInput::const_iterator` formula)  
*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` & \_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` & \_formula, `FormulasT` & \_origins) const
- void `getOrigins` (const `FormulaT` & \_formula, `FormulaSetT` & \_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` & \_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> & \_origins)
- void `informBackends` (const `FormulaT` & \_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` & \_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*

- std::pair< [ModuleInput::iterator](#), bool > [addReceivedSubformulaToPassedFormula](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool [originInReceivedFormula](#) (const [FormulaT](#) &\_origin) const
  - std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
  - std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
 

*Adds the given formula to the passed formula.*
  - std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
  - void [getInfeasibleSubsets](#) ()
 

*Copies the infeasible subsets of the passed formula.*
  - std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
  - const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - void [getBackendsModel](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
  - void [getBackendsAllModels](#) () const
 

*Runs the backend solvers on the passed formula.*
  - virtual [Answer](#) [runBackends](#) (bool \_final, bool \_full, carl::Variable \_objective)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
  - virtual [Answer](#) [runBackends](#) ()
 

*Merges the two vectors of sets into the first one.*
  - size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
  - bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
  - bool [branchAt](#) (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*branchAt (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())*
  - bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*branchAt (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())*
  - bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*branchAt (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())*

- void `splitUnequalConstraint` (const `FormulaT` &)  
*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &`formula`)  
*Adds a formula to the `InformationRelevantFormula`.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()  
*Gets all `InformationRelevantFormulas`.*
- bool `isLemmaLevel` (`LemmaLevel` `level`)  
*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &`_modelA`, const `Model` &`_modelB`)  
*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`  
*A reference to the manager.*
- `Model` `mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`  
*Stores all satisfying assignments.*
- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals `mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`  
*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput`::iterator `mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*

- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.100.1 Member Enumeration Documentation

#### 0.15.100.1.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.100.2 Member Function Documentation

#### 0.15.100.2.1 add() `bool smtrat::Module::add ( ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

#### 0.15.100.2.2 addConstraintToInform() `void smtrat::Module::addConstraintToInform ( const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in `smtrat::SATModule< Settings >`.

#### 0.15.100.2.3 addCore() `virtual bool smtrat::Module::addCore ( ModuleInput::const_iterator formula ) [inline], [protected], [virtual], [inherited]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>formula</i> | The sub-formula to take additionally into account. |
|----------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.100.2.4 addInformationRelevantFormula()** void smrat::Module::addInformationRelevantFormula (  
 const *FormulaT* & *formula*) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.100.2.5 addLemma()** void smrat::Module::addLemma (  
 const *FormulaT* & *\_lemma*,  
 const *LemmaType* & *\_lt* = *LemmaType::NORMAL*,  
 const *FormulaT* & *\_preferredFormula* = *FormulaT( carl::FormulaType::TRUE )*) [inline],  
[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.100.2.6 addOrigin()** void smrat::Module::addOrigin (  
*ModuleInput::iterator* *\_formula*,  
 const *FormulaT* & *\_origin*) [inline], [protected], [inherited]

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

```
0.15.100.2.7 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smrat::Module::addReceivedSubformulaToPassedFormula (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

```
0.15.100.2.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smrat<-
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.100.2.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smrat<-
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.100.2.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
```

```
const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
Adds the given formula to the passed formula.
```

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.100.2.11 `allModels()`** const std::vector<[Model](#)>& smtrat::Module::allModels () const [inline], [inherited]

**Returns**

All satisfying assignments, if existent.

**0.15.100.2.12 `anAnswerFound()`** bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.100.2.13 `answerFound()`** const [smtrat::Conditionals](#)& smtrat::Module::answerFound () const [inline], [inherited]

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.100.2.14 `backendsModel()`** const [Model](#) & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.100.2.15 `branchAt()` [1/4]** bool smtrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector< FormulaT >()) [inline],
[protected], [inherited]
```

```
0.15.100.2.16 branchAt() [2/4] bool smtrat::Module::branchAt (
 carl::Variable::Arg _var,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

```
0.15.100.2.17 branchAt() [3/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

```
0.15.100.2.18 branchAt() [4/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.100.2.19 check() Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::PModule](#).

### 0.15.100.2.20 `checkCore()` Answer [smrat::Module::checkCore](#) ( ) [protected], [virtual], [inherited]

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::SymmetryModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::SplitSOSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::MCBModule< Settings >](#), [smrat::LVEModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::ICEModule< Settings >](#), [smrat::GBPPModule< Settings >](#), [smrat::GBModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::ESModule< Settings >](#), [smrat::EQPreprocessingModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::EMModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::CNFerModule](#), [smrat::BVMModule< Settings >](#), and [smrat::BEModule< Settings >](#).

### 0.15.100.2.21 `checkInfeasibleSubsetForMinimality()` void [smrat::Module::checkInfeasibleSubsetForMinimality](#) ( `std::vector< FormulaSetT >::const_iterator _infssubset,` `const std::string & _filename = "smaller_muses",` `unsigned _maxSizeDifference = 1` ) const [inherited]

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

### 0.15.100.2.22 `checkModel()` unsigned [smrat::Module::checkModel](#) ( ) const [protected], [inherited]

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.100.2.23 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.100.2.24 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.100.2.25 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.100.2.26 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.100.2.27 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.100.2.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.100.2.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.100.2.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.100.2.31 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.100.2.32 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.100.2.33 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.100.2.34 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.100.2.35 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.100.2.36 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.100.2.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.100.2.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.100.2.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.100.2.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const` [inline], [inherited]

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.100.2.41 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const` [inline], [inherited]

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.100.2.42 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.100.2.43 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.100.2.44 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.100.2.45 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.100.2.46 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.100.2.47 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.100.2.48 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.100.2.49 getModelEqualities()** std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.100.2.50 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.100.2.51 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.100.2.52 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

#### Parameters

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.100.2.53 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.100.2.54 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.100.2.55 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

#### Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.100.2.56 `id()`** `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.100.2.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.100.2.58 `inform()`** `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.100.2.59 `informBackends()`** `void smtrat::Module::informBackends ( const FormulaT & _constraint ) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.100.2.60 `informCore()`** `virtual bool smtrat::Module::informCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.100.2.61 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

#### Returns

The position of the first constraint of which no backend has been informed about.

**0.15.100.2.62 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.100.2.63 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.100.2.64 isLemmaLevel()** `bool smrat::Module::isLemmaLevel( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.100.2.65 isPreprocessor()** `bool smrat::Module::isPreprocessor( ) const [inline], [inherited]`

#### Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.100.2.66 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas( ) const [inline], [inherited]`

#### Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.100.2.67 merge()** `std::vector< FormulaT > smrat::Module::merge( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.100.2.68 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.100.2.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.100.2.70 `moduleName()`** `virtual std::string smtrat::Module::moduleName () const [inline], [virtual], [inherited]`**Returns**

The name of the given module type as name.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SplitSOSModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LVEModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::GBModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQPreprocessingModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::BEModule< Settings >](#).

**0.15.100.2.71 `objective()`** `carl::Variable smtrat::Module::objective () const [inline], [inherited]`**0.15.100.2.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (const FormulaT & _origin ) const [protected], [inherited]`

**0.15.100.2.73 passedFormulaBegin()** `ModuleInput::iterator smrat::Module::passedFormulaBegin ( )` [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.100.2.74 passedFormulaEnd()** `ModuleInput::iterator smrat::Module::passedFormulaEnd ( )` [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.100.2.75 pPassedFormula()** `const ModuleInput* smrat::Module::pPassedFormula ( ) const` [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.100.2.76 pReceivedFormula()** `const ModuleInput* smrat::Module::pReceivedFormula ( ) const` [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.100.2.77 print()** `void smrat::Module::print ( const std::string & _initiation = "***" ) const` [inherited]

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.100.2.78 printAllModels()** `void smrat::Module::printAllModels ( std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.100.2.79 printInfeasibleSubsets()** `void smrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

## Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.100.2.80 `printModel()`** `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

## Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.100.2.81 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

## Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.100.2.82 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

## Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.100.2.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

## Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

## Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.100.2.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.100.2.85 `receivedFormulasAsInfeasibleSubset()`** void smtrat::Module::receivedFormulasAsInfeasibleSubset (

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

**0.15.100.2.86 `receivedVariable()`** bool smtrat::Module::receivedVariable (

```
 carl::Variable::Arg _var) const [inline], [inherited]
```

**0.15.100.2.87 `remove()`** void smtrat::Module::remove (

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.100.2.88 `removeCore()`** virtual void smtrat::Module::removeCore (

```
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICEModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::GBModule< Settings >](#).

**0.15.100.2.89 `removeOrigin()`** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.100.2.90 `removeOrigins()`** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (

```
ModuleInput::iterator _formula,
const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

#### 0.15.100.2.91 **rPassedFormula()** const ModuleInput& smtrat::Module::rPassedFormula () const [inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

#### 0.15.100.2.92 **rReceivedFormula()** const ModuleInput& smtrat::Module::rReceivedFormula () const [inline], [inherited]

**Returns**

A reference to the formula passed to this module.

#### 0.15.100.2.93 **runBackends()** [1/2] virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]

Reimplemented in [smtrat::PModule](#).

#### 0.15.100.2.94 **runBackends()** [2/2] Answer smtrat::Module::runBackends (

```
bool _final,
bool _full,
carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

#### 0.15.100.2.95 **setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]

Sets this modules unique ID to identify itself.

**Parameters**

|                             |                                    |
|-----------------------------|------------------------------------|
| $\leftrightarrow$<br>$\_id$ | The id to set this module's id to. |
|-----------------------------|------------------------------------|

**0.15.100.2.96 `setThreadPriority()`** `void smtrat::Module::setThreadPriority (`  
    `thread_priority _threadPriority )` [inline], [inherited]  
Sets the priority of this module to get a thread for running its check procedure.

Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.100.2.97 `solverState()`** `Answer smtrat::Module::solverState ( ) const` [inline], [inherited]

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.100.2.98 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint (`  
    `const FormulaT & _unequalConstraint )` [protected], [inherited]

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.100.2.99 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const` [inline], [inherited]

Returns

The priority of this module to get a thread for running its check procedure.

**0.15.100.2.100 `updateAllModels()`** `void smtrat::Module::updateAllModels ( )` [virtual], [inherited]

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.100.2.101 `updateLemmas()`** `void smtrat::Module::updateLemmas ( )` [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.100.2.102 `updateModel()`** `void smtrat::Module::updateModel ( ) const` [virtual], [inherited]

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::PModule](#), [smtrat::VSModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::MCBModule< Settings >](#), [smtrat::LVEModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#),

`smrat::IncWidthModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, smrat::ICPModule< ICPSettings1 >, smrat::GBPPModule< Settings >, smrat::FPPModule< Settings >, smrat::FouMoModule< Settings >, smrat::ESModule< Settings >, smrat::EQPreprocessingModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::CubeLIAModule< Settings >, smrat::CSplitModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.100.2.103 `usedBackends()`** `const std::vector<Module*>& smrat::Module::usedBackends( )`  
`const [inline], [inherited]`

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.100.3 Field Documentation

**0.15.100.3.1 `mAllBackends`** `std::vector<Module*> smrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.100.3.2 `mAllModels`** `std::vector<Model> smrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

**0.15.100.3.3 `mBackendsFoundAnswer`** `std::atomic_bool* smrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.100.3.4 `mConstraintsToInform`** `carl::FastSet<FormulaT> smrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

**0.15.100.3.5 `mFinalCheck`** `bool smrat::Module::mFinalCheck [protected], [inherited]`  
 true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.100.3.6 `mFirstPosInLastBranches`** `std::size_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

**0.15.100.3.7 `mFirstSubformulaToPass`** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.100.3.8 `mFirstUncheckedReceivedSubformula`** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.100.3.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.100.3.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.100.3.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.100.3.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.100.3.13 mLastBranches** `std::vector<Branching>` `smtrat::Module::mLastBranches` = `std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.100.3.14 mLemmas** `std::vector<Lemma>` `smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.100.3.15 mModel** `Model` `smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.100.3.16 mModelComputed** `bool` `smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.100.3.17 mNumOfBranchVarsToStore** `std::size_t` `smtrat::Module::mNumOfBranchVarsToStore` = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.100.3.18 mObjectiveVariable** `carl::Variable` `smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.100.3.19 mOldSplittingVariables** `std::vector<FormulaT>` `smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.100.3.20 mpManager** `Manager*` `const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.100.3.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter`  
 [mutable], [protected], [inherited]  
 Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.100.3.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected],  
 [inherited]  
 States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.100.3.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheory←Propagations` [protected], [inherited]

**0.15.100.3.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends` [protected],  
 [inherited]  
 The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.100.3.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters`  
 [protected], [inherited]  
 Maps variables to the number of their occurrences.

## 0.15.101 smtrat::CollectBoolsInUEQs Struct Reference

```
#include <BoolUEQRewriter.h>
```

### Public Member Functions

- void `visit_ueq` (const `FormulaT` &formula)
- `CollectBoolsInUEQs` (const `CollectBoolsInUEQs` &)=delete
- `CollectBoolsInUEQs & operator=` (const `CollectBoolsInUEQs` &)=delete
- `CollectBoolsInUEQs ()=default`
- `CollectBoolsInUEQs (CollectBoolsInUEQs &&)=default`
- `CollectBoolsInUEQs & operator= (CollectBoolsInUEQs &&)=default`
- bool `foundBoolInUEQ ()`

### Friends

- struct `BoolUEQRewriter`

### 0.15.101.1 Constructor & Destructor Documentation

**0.15.101.1.1 CollectBoolsInUEQs()** [1/3] `smtrat::CollectBoolsInUEQs::CollectBoolsInUEQs (`  
`const CollectBoolsInUEQs & )` [delete]

**0.15.101.1.2 CollectBoolsInUEQs()** [2/3] `smtrat::CollectBoolsInUEQs::CollectBoolsInUEQs ( )`  
 [default]

**0.15.101.1.3 CollectBoolsInUEQs()** [3/3] `smtrat::CollectBoolsInUEQs::CollectBoolsInUEQs (`  
`CollectBoolsInUEQs && )` [default]

### 0.15.101.2 Member Function Documentation

**0.15.101.2.1 `foundBoolInUEQ()`** `bool smtrat::CollectBoolsInUEQs::foundBoolInUEQ ( ) [inline]`

**0.15.101.2.2 `operator=()` [1/2]** `CollectBoolsInUEQs& smtrat::CollectBoolsInUEQs::operator= ( CollectBoolsInUEQs && ) [default]`

**0.15.101.2.3 `operator=()` [2/2]** `CollectBoolsInUEQs& smtrat::CollectBoolsInUEQs::operator= ( const CollectBoolsInUEQs & ) [delete]`

**0.15.101.2.4 `visit_ueq()`** `void smtrat::CollectBoolsInUEQs::visit_ueq ( const FormulaT & formula ) [inline]`

### 0.15.101.3 Friends And Related Function Documentation

**0.15.101.3.1 `BoolUEQRewriter`** `friend struct BoolUEQRewriter [friend]`

## 0.15.102 `smtrat::CollectionWithOrigins< Element, Origin >` Class Template Reference

#include <IntBlastModule.h>

### Public Types

- `typedef Super::iterator iterator`
- `typedef Super::const_iterator const_iterator`

### Public Member Functions

- `CollectionWithOrigins (bool _trackElementsWithoutOrigins=true)`
- `bool contains (const Element &_element)`
- `bool add (const Element &_element, const Origin &_origin)`
- `bool removeOrigin (const Origin &_origin)`
- `bool removeOrigins (const std::set< Origin > &_origins)`
- `iterator begin ()`
- `iterator end ()`
- `const_iterator cbegin () const`
- `const_iterator cend () const`
- `const std::set< Element > & elementsWithoutOrigins () const`
- `void clearElementsWithoutOrigins ()`

### 0.15.102.1 Member Typedef Documentation

**0.15.102.1.1 `const_iterator`** `template<typename Element , typename Origin > typedef Super::const_iterator smtrat::CollectionWithOrigins< Element, Origin >::const_iterator`

**0.15.102.1.2 `iterator`** `template<typename Element , typename Origin > typedef Super::iterator smtrat::CollectionWithOrigins< Element, Origin >::iterator`

### 0.15.102.2 Constructor & Destructor Documentation

**0.15.102.2.1 CollectionWithOrigins()** template<typename Element , typename Origin >  
smtrat::CollectionWithOrigins< Element, Origin >::CollectionWithOrigins (

```
 bool _trackElementsWithoutOrigins = true) [inline]
```

### 0.15.102.3 Member Function Documentation

**0.15.102.3.1 add()** template<typename Element , typename Origin >  
bool smtrat::CollectionWithOrigins< Element, Origin >::add (

```
 const Element & _element,
 const Origin & _origin) [inline]
```

**0.15.102.3.2 begin()** template<typename Element , typename Origin >  
iterator smtrat::CollectionWithOrigins< Element, Origin >::begin ( ) [inline]

**0.15.102.3.3 cbegin()** template<typename Element , typename Origin >  
const\_iterator smtrat::CollectionWithOrigins< Element, Origin >::cbegin ( ) const [inline]

**0.15.102.3.4 cend()** template<typename Element , typename Origin >  
const\_iterator smtrat::CollectionWithOrigins< Element, Origin >::cend ( ) const [inline]

**0.15.102.3.5 clearElementsWithoutOrigins()** template<typename Element , typename Origin >  
void smtrat::CollectionWithOrigins< Element, Origin >::clearElementsWithoutOrigins ( ) [inline]

**0.15.102.3.6 contains()** template<typename Element , typename Origin >  
bool smtrat::CollectionWithOrigins< Element, Origin >::contains (

```
 const Element & _element) [inline]
```

**0.15.102.3.7 elementsWithoutOrigins()** template<typename Element , typename Origin >  
const std::set<Element>& smtrat::CollectionWithOrigins< Element, Origin >::elementsWithoutOrigins ( ) const [inline]

**0.15.102.3.8 end()** template<typename Element , typename Origin >  
iterator smtrat::CollectionWithOrigins< Element, Origin >::end ( ) [inline]

**0.15.102.3.9 removeOrigin()** template<typename Element , typename Origin >  
bool smtrat::CollectionWithOrigins< Element, Origin >::removeOrigin (

```
 const Origin & _origin) [inline]
```

**0.15.102.3.10 removeOrigins()** template<typename Element , typename Origin >  
bool smtrat::CollectionWithOrigins< Element, Origin >::removeOrigins (

```
 const std::set< Origin > & _origins) [inline]
```

### 0.15.103 smtrat::ICPModule< Settings >::comp Struct Reference

TypeDefs:

```
#include <ICPModule.h>
```

#### Public Member Functions

- bool `operator()` (std::pair< double, unsigned > lhs, std::pair< double, unsigned > rhs) const

#### 0.15.103.1 Detailed Description

```
template<class Settings>
struct smtrat::ICPModule< Settings >::comp
```

TypeDefs:

#### 0.15.103.2 Member Function Documentation

```
0.15.103.2.1 operator()() template<class Settings >
bool smtrat::ICPModule< Settings >::comp::operator() (
 std::pair< double, unsigned > lhs,
 std::pair< double, unsigned > rhs) const [inline]
```

### 0.15.104 smtrat::cadcells::datastructures::CompoundMax Struct Reference

Represents the maximum function of the contained indexed root functions.

```
#include <roots.h>
```

#### Public Member Functions

- void `polys` (boost::container::flat\_set< PolyRef > &result) const

#### Data Fields

- std::vector< IndexedRoot > roots

#### 0.15.104.1 Detailed Description

Represents the maximum function of the contained indexed root functions.

#### 0.15.104.2 Member Function Documentation

```
0.15.104.2.1 polys() void smtrat::cadcells::datastructures::CompoundMax::polys (
 boost::container::flat_set< PolyRef > & result) const [inline]
```

#### 0.15.104.3 Field Documentation

```
0.15.104.3.1 roots std::vector<IndexedRoot> smtrat::cadcells::datastructures::CompoundMax::roots
```

### 0.15.105 smtrat::cadcells::datastructures::CompoundMin Struct Reference

Represents the minimum function of the contained indexed root functions.

```
#include <roots.h>
```

## Public Member Functions

- void `polys` (boost::container::flat\_set< `PolyRef` > &result) const

## Data Fields

- std::vector< `IndexedRoot` > `roots`

### 0.15.105.1 Detailed Description

Represents the minimum function of the contained indexed root functions.

### 0.15.105.2 Member Function Documentation

#### 0.15.105.2.1 `polys()` void smtrat::cadcells::datastructures::CompoundMin::polys ( boost::container::flat\_set< `PolyRef` > & result ) const [inline]

#### 0.15.105.3 Field Documentation

#### 0.15.105.3.1 `roots` std::vector<`IndexedRoot`> smtrat::cadcells::datastructures::CompoundMin::roots

## 0.15.106 smtrat::vs::Condition Class Reference

```
#include <Condition.h>
```

## Public Member Functions

- `Condition` (const `smtrat::ConstraintT` &, size\_t \_id, size\_t=0, bool=false, const carl::PointerSet< `Condition` > &=carl::PointerSet< `Condition` >(), bool=false)
- `Condition` (const `Condition` &, size\_t \_id)
- `Condition` (const `Condition` &)=delete
- `~Condition` ()
- bool & `rFlag` () const
- bool `flag` () const
- bool & `rRecentlyAdded` () const
- bool `recentlyAdded` () const
- size\_t & `rValuation` () const
- size\_t `valuation` () const
- size\_t `id` () const
- const `smtrat::ConstraintT` & `constraint` () const
- carl::PointerSet< `Condition` > \* `pOriginalConditions` () const
- const carl::PointerSet< `Condition` > & `originalConditions` () const
- double `valuate` (const carl::Variable &, size\_t, bool) const

*Valuates the constraint according to a variable (it possibly not contains).*

- bool `operator==` (const `Condition` &) const

*Checks the equality of a given condition (right hand side) with this condition (left hand side).*

- bool `operator<` (const `Condition` &) const

*Checks if the given condition (right hand side) is greater than this condition (left hand side).*

- void `print` (std::ostream &=std::cout) const

*Prints the condition to an output stream.*

**Friends**

- std::ostream & `operator<<` (std::ostream &\_out, const `Condition` &\_condition)

**0.15.106.1 Constructor & Destructor Documentation****0.15.106.1.1 Condition() [1/3]** smtrat::vs::Condition::Condition (

```
 const smtrat::ConstraintT & _cons,
 size_t _id,
 size_t _val = 0,
 bool _flag = false,
 const carl::PointerSet< Condition > & _oConds = carl::PointerSet<Condition>(),
 bool _rAdded = false)
```

**0.15.106.1.2 Condition() [2/3]** smtrat::vs::Condition::Condition (

```
 const Condition & _cond,
 size_t _id)
```

**0.15.106.1.3 Condition() [3/3]** smtrat::vs::Condition::Condition (

```
 const Condition &) [delete]
```

**0.15.106.1.4 ~Condition()** smtrat::vs::Condition::~Condition ( )**0.15.106.2 Member Function Documentation****0.15.106.2.1 constraint()** const smtrat::ConstraintT& smtrat::vs::Condition::constraint ( ) const [inline]**0.15.106.2.2 flag()** bool smtrat::vs::Condition::flag ( ) const [inline]**0.15.106.2.3 id()** size\_t smtrat::vs::Condition::id ( ) const [inline]**0.15.106.2.4 operator<()** bool smtrat::vs::Condition::operator< (

```
 const Condition & _condition) const
```

Checks if the given condition (right hand side) is greater than this condition (left hand side).

**Parameters**

|                         |                                      |
|-------------------------|--------------------------------------|
| <code>_condition</code> | The condition to compare with (rhs). |
|-------------------------|--------------------------------------|

**Returns**

true ,if the given substitution is greater than this substitution; false ,otherwise.

**0.15.106.2.5 operator==()** bool smtrat::vs::Condition::operator== (

```
 const Condition & _condition) const
```

Checks the equality of a given condition (right hand side) with this condition (left hand side).

#### Parameters

|                         |                                      |
|-------------------------|--------------------------------------|
| <code>_condition</code> | The condition to compare with (rhs). |
|-------------------------|--------------------------------------|

#### Returns

`true`, if the given condition is equal to this condition; `false`, otherwise.

**0.15.106.2.6 `originalConditions()`** `const carl::PointerSet<Condition>& smtrat::vs::Condition::originalConditions ( ) const [inline]`

**0.15.106.2.7 `pOriginalConditions()`** `carl::PointerSet<Condition>* smtrat::vs::Condition::pOriginalConditions ( ) const [inline]`

**0.15.106.2.8 `print()`** `void smtrat::vs::Condition::print ( std::ostream & _out = std::cout ) const`

Prints the condition to an output stream.

#### Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <code>_out</code> | The output stream, where it should print. |
|-------------------|-------------------------------------------|

**0.15.106.2.9 `recentlyAdded()`** `bool smtrat::vs::Condition::recentlyAdded ( ) const [inline]`

**0.15.106.2.10 `rFlag()`** `bool& smtrat::vs::Condition::rFlag ( ) const [inline]`

**0.15.106.2.11 `rRecentlyAdded()`** `bool& smtrat::vs::Condition::rRecentlyAdded ( ) const [inline]`

**0.15.106.2.12 `rValuation()`** `size_t& smtrat::vs::Condition::rValuation ( ) const [inline]`

**0.15.106.2.13 `valuate()`** `double smtrat::vs::Condition::valuate ( const carl::Variable & _consideredVariable, size_t _maxNumberOfVars, bool _preferEquation ) const`

Valuates the constraint according to a variable (it possibly not contains).

#### Parameters

|                                  |                                                     |
|----------------------------------|-----------------------------------------------------|
| <code>_consideredVariable</code> | The variable which is considered in this valuation. |
| <code>_maxNumberOfVars</code>    |                                                     |

**Returns**

A valuation of the constraint according to an heuristic.

**0.15.106.2.14 `valuation()`** `size_t smtrat::vs::Condition::valuation () const [inline]`**0.15.106.3 Friends And Related Function Documentation****0.15.106.3.1 `operator<<`** `std::ostream& operator<< ( std::ostream & _out, const Condition & _condition ) [friend]`**0.15.107 `benchmax::CondorBackend` Class Reference**

[Backend](#) for the HTCondor batch system.

```
#include <CondorBackend.h>
```

**Public Member Functions**

- void `run` (const [Jobs](#) &, bool)  
*Run all tools on all benchmarks.*
- bool `suspendable` () const
- void `process_results` (const [Jobs](#) &jobs, bool check\_finished)
- void `addResult` (const [Tool](#) \*tool, const [fs::path](#) &file, [BenchmarkResult](#) &&result)  
*Add a result.*

**Protected Member Functions**

- virtual void `execute` (const [Tool](#) \*, const [fs::path](#) &)  
*No-op version of execute.*
- virtual void `startTool` (const [Tool](#) \*)  
*Hook for every tool at the beginning.*
- virtual void `finalize` ()  
*Hook to allow for asynchronous backends to wait for jobs to terminate.*
- void `madeProgress` ([std::size\\_t](#) files=1)  
*Can be called to give information about the current progress, if available.*
- virtual bool `collect_results` (const [Jobs](#) &, bool)
- void `sanitize_results` (const [Jobs](#) &jobs) const
- void `write_results` (const [Jobs](#) &jobs) const

**Protected Attributes**

- [std::size\\_t](#) `mExpectedJobs`  
*Number of jobs that should be run.*
- [std::atomic< std::size\\_t >](#) `mFinishedJobs`  
*Number of jobs that are finished.*
- [std::atomic< std::size\\_t >](#) `mLastPercent`  
*Percentage of finished jobs when `madeProgress()` was last called.*

**0.15.107.1 Detailed Description**

[Backend](#) for the HTCondor batch system.

Currently submits all jobs individually and asynchronously waits for them to finish.

### 0.15.107.2 Member Function Documentation

**0.15.107.2.1 `addResult()`** void benchmax::Backend::addResult ( const `Tool` \* `tool`, const `fs::path` & `file`, `BenchmarkResult` && `result` ) [inline], [inherited]

Add a result.

**0.15.107.2.2 `collect_results()`** virtual bool benchmax::Backend::collect\_results ( const `Jobs` & , bool ) [inline], [protected], [virtual], [inherited]

**0.15.107.2.3 `execute()`** virtual void benchmax::CondorBackend::execute ( const `Tool` \* , const `fs::path` & ) [inline], [protected], [virtual]

No-op version of `execute`.

Reimplemented from `benchmax::Backend`.

**0.15.107.2.4 `finalize()`** virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual], [inherited]

Hook to allow for asynchronous backends to wait for jobs to terminate.

**0.15.107.2.5 `madeProgress()`** void benchmax::Backend::madeProgress ( std::size\_t `files` = 1 ) [inline], [protected], [inherited]

Can be called to give information about the current progress, if available.

**0.15.107.2.6 `process_results()`** void benchmax::Backend::process\_results ( const `Jobs` & `jobs`, bool `check_finished` ) [inline], [inherited]

**0.15.107.2.7 `run()`** void benchmax::CondorBackend::run ( const `Jobs` & , bool ) [inline]

Run all tools on all benchmarks.

**0.15.107.2.8 `sanitize_results()`** void benchmax::Backend::sanitize\_results ( const `Jobs` & `jobs` ) const [inline], [protected], [inherited]

**0.15.107.2.9 `startTool()`** virtual void benchmax::Backend::startTool ( const `Tool` \* ) [inline], [protected], [virtual], [inherited]

Hook for every tool at the beginning.

Can be used to upload the tool to some remote system.

**0.15.107.2.10 `suspendable()`** bool benchmax::Backend::suspendable ( ) const [inline], [inherited]

```
0.15.107.2.11 write_results() void benchmax::Backend::write_results (
 const Jobs & jobs) const [inline], [protected], [inherited]
```

### 0.15.107.3 Field Documentation

**0.15.107.3.1 mExpectedJobs** std::size\_t benchmax::Backend::mExpectedJobs [protected], [inherited]  
Number of jobs that should be run.

**0.15.107.3.2 mFinishedJobs** std::atomic<std::size\_t> benchmax::Backend::mFinishedJobs [protected], [inherited]  
Number of jobs that are finished.

**0.15.107.3.3 mLastPercent** std::atomic<std::size\_t> benchmax::Backend::mLastPercent [protected], [inherited]  
Percentage of finished jobs when [madeProgress\(\)](#) was last called.

## 0.15.108 smrat::mcsat::fm::ConflictGenerator< Comparator > Struct Template Reference

```
#include <ConflictGenerator.h>
```

### Public Member Functions

- [ConflictGenerator](#) (const std::vector< [ConstraintT](#) > &bounds, const [Model](#) &m, carl::Variable v)
- template<typename Callback >  
void [generateExplanation](#) (Callback &&callback)

### 0.15.108.1 Constructor & Destructor Documentation

```
0.15.108.1.1 ConflictGenerator() template<class Comparator >
smrat::mcsat::fm::ConflictGenerator< Comparator >::ConflictGenerator (
 const std::vector< ConstraintT > & bounds,
 const Model & m,
 carl::Variable v) [inline]
```

### 0.15.108.2 Member Function Documentation

```
0.15.108.2.1 generateExplanation() template<class Comparator >
template<typename Callback >
void smrat::mcsat::fm::ConflictGenerator< Comparator >::generateExplanation (
 Callback && callback) [inline]
```

## 0.15.109 smrat::cad::ConflictGraph Class Reference

Representation of a bipartite graph (C+S, E) of vertices C and S, representing the constraints and samples, respectively.

```
#include <ConflictGraph.h>
```

## Public Member Functions

- `ConflictGraph (std::size_t constraints)`  
*Retrieves the constraint that covers the most samples.*
- `std::size_t coveredSamples (std::size_t id) const`
- `void addSample (const Sample &sample)`
- `std::size_t getMaxDegreeConstraint () const`  
*Removes the given constraint and disable all sample points covered by this constraint.*
- `std::vector< size_t > selectEssentialConstraints ()`  
*Selects all essential constraints and returns their indices.*
- `ConflictGraph removeDuplicateColumns ()`  
*Returns a new `ConflictGraph` whose adjacency matrix consists only of the unique columns of the adjacency matrix of this graph.*
- `bool hasRemainingSamples () const`  
*Checks if there are samples still uncovered.*
- `std::size_t numSamples () const`
- `std::size_t numRemainingConstraints () const`
- `carl::Bitset getData (std::size_t id)`
- `std::vector< std::pair< std::size_t, carl::Bitset > > getRemainingConstraints ()`

## Friends

- `std::ostream & operator<< (std::ostream &os, const ConflictGraph &cg)`

### 0.15.109.1 Detailed Description

Representation of a bipartite graph ( $C + S, E$ ) of vertices  $C$  and  $S$ , representing the constraints and samples, respectively.

A vertex from  $C$  and a vertex from  $S$  are connected by an edge in  $E$  iff the corresponding constraint conflicts with the corresponding sample point.

### 0.15.109.2 Constructor & Destructor Documentation

**0.15.109.2.1 `ConflictGraph()`** `smtrat::cad::ConflictGraph::ConflictGraph ( std::size_t constraints ) [inline]`

### 0.15.109.3 Member Function Documentation

**0.15.109.3.1 `addSample()`** `void smtrat::cad::ConflictGraph::addSample ( const Sample & sample )`

**0.15.109.3.2 `coveredSamples()`** `std::size_t smtrat::cad::ConflictGraph::coveredSamples ( std::size_t id ) const [inline]`

**0.15.109.3.3 `getData()`** `carl::Bitset smtrat::cad::ConflictGraph::getData ( std::size_t id )`

**0.15.109.3.4 `getMaxDegreeConstraint()`** `std::size_t smtrat::cad::ConflictGraph::getMaxDegreeConstraint ( ) const`  
Retrieves the constraint that covers the most samples.

**0.15.109.3.5 `getRemainingConstraints()`** `std::vector< std::pair< std::size_t, carl::Bitset > > smtrat::cad::ConflictGraph::getRemainingConstraints ( )`

**0.15.109.3.6 `hasRemainingSamples()`** `bool smtrat::cad::ConflictGraph::hasRemainingSamples ( ) const`  
Checks if there are samples still uncovered.

**0.15.109.3.7 `numRemainingConstraints()`** `std::size_t smtrat::cad::ConflictGraph::numRemainingConstraints ( ) const`

**0.15.109.3.8 `numSamples()`** `std::size_t smtrat::cad::ConflictGraph::numSamples ( ) const`

**0.15.109.3.9 `removeDuplicateColumns()`** `ConflictGraph smtrat::cad::ConflictGraph::removeDuplicateColumns ( )`  
Returns a new `ConflictGraph` whose adjacency matrix consists only of the unique columns of the adjacency matrix of this graph.

**0.15.109.3.10 `selectConstraint()`** `void smtrat::cad::ConflictGraph::selectConstraint ( std::size_t id )`

Removes the given constraint and disable all sample points covered by this constraint.

**0.15.109.3.11 `selectEssentialConstraints()`** `std::vector< std::size_t > smtrat::cad::ConflictGraph::selectEssentialConstraints ( )`  
Selects all essential constraints and returns their indices.

## 0.15.109.4 Friends And Related Function Documentation

**0.15.109.4.1 `operator<<`** `std::ostream& operator<< ( std::ostream & os, const ConflictGraph & cg ) [friend]`

## 0.15.110 `smtrat::parser::Theories::ConstantAdder` Struct Reference

Helper functor for `addConstants()` method.

```
#include <Theories.h>
```

### Public Member Functions

- `ConstantAdder (qi::symbols< char, types::ConstType > &constants)`
- template<typename T>  
`void operator() (T *)`

**Data Fields**

- `qi::symbols< char, types::ConstType > & constants`

**0.15.110.1 Detailed Description**

Helper functor for `addConstants()` method.

**0.15.110.2 Constructor & Destructor Documentation**

**0.15.110.2.1 ConstantAdder()** `smtrat::parser::Theories::ConstantAdder::ConstantAdder ( qi::symbols< char, types::ConstType > & constants ) [inline]`

**0.15.110.3 Member Function Documentation**

**0.15.110.3.1 operator()()** `template<typename T > void smtrat::parser::Theories::ConstantAdder::operator() ( T * ) [inline]`

**0.15.110.4 Field Documentation**

**0.15.110.4.1 constants** `qi::symbols<char, types::ConstType>& smtrat::parser::Theories::ConstantAdder::constants`

**0.15.111 smtrat::cad::CADConstraints< BT >::ConstraintComparator Struct Reference**

```
#include <CADConstraints.h>
```

**Public Member Functions**

- `std::size_t complexity (const ConstraintT &c) const`
- `bool operator() (const ConstraintT &lhs, const ConstraintT &rhs) const`

**0.15.111.1 Member Function Documentation**

**0.15.111.1.1 complexity()** `template<Backtracking BT> std::size_t smtrat::cad::CADConstraints< BT >::ConstraintComparator::complexity ( const ConstraintT & c ) const [inline]`

**0.15.111.1.2 operator()()** `template<Backtracking BT> bool smtrat::cad::CADConstraints< BT >::ConstraintComparator::operator() ( const ConstraintT & lhs, const ConstraintT & rhs ) const [inline]`

**0.15.112 smtrat::cad::preprocessor::ConstraintUpdate Struct Reference**

```
#include <CADPreprocessor.h>
```

**Data Fields**

- std::vector< ConstraintT > toAdd
- std::vector< ConstraintT > toRemove

**0.15.112.1 Field Documentation**

**0.15.112.1.1 toAdd** std::vector< ConstraintT > smtrat::cad::preprocessor::ConstraintUpdate<  
::toAdd

**0.15.112.1.2 toRemove** std::vector< ConstraintT > smtrat::cad::preprocessor::ConstraintUpdate<  
::toRemove

**0.15.113 smtrat::ConstrTree Class Reference**

```
#include <IntBlastModule.h>
```

**Public Member Functions**

- ConstrTree (const ConstraintT &\_constraint)
- ~ConstrTree ()
- carl::Relation relation () const
- const PolyTree & left () const
- const PolyTree & right () const
- const ConstraintT & constraint () const

**0.15.113.1 Constructor & Destructor Documentation**

**0.15.113.1.1 ConstrTree()** smtrat::ConstrTree::ConstrTree (const ConstraintT & \_constraint ) [inline]

**0.15.113.1.2 ~ConstrTree()** smtrat::ConstrTree::~ConstrTree ( ) [inline]

**0.15.113.2 Member Function Documentation**

**0.15.113.2.1 constraint()** const ConstraintT& smtrat::ConstrTree::constraint ( ) const [inline]

**0.15.113.2.2 left()** const PolyTree& smtrat::ConstrTree::left ( ) const [inline]

**0.15.113.2.3 relation()** carl::Relation smtrat::ConstrTree::relation ( ) const [inline]

**0.15.113.2.4 right()** const PolyTree& smtrat::ConstrTree::right ( ) const [inline]

### 0.15.114 delta::Consumer Class Reference

This class takes care of asynchronous execution of calls to the solver.

```
#include <Consumer.h>
```

#### Public Member Functions

- `Consumer` (const std::string &tempPrefix, std::size\_t threads, const `Checker` &checker)  
*Constructor.*
- `~Consumer` ()  
*Destructor.*
- void `consume` (const `Node` &n, const std::string &message, std::size\_t num)  
*Initiate asynchronous check for the given node.*
- bool `wait` ()  
*Wait for at least one job to finish.*
- void `reset` ()  
*Reset this executor.*
- bool `hasResult` () const  
*Checks if at least one job was successful.*
- std::tuple< `Node`, std::string, std::size\_t > `getResult` ()  
*Returns the result, assuming that one exists.*
- std::pair< unsigned, unsigned > `getProgress` () const  
*Returns the current progress.*

#### 0.15.114.1 Detailed Description

This class takes care of asynchronous execution of calls to the solver.

#### 0.15.114.2 Constructor & Destructor Documentation

**0.15.114.2.1 Consumer()** `delta::Consumer::Consumer (`  
`const std::string & tempPrefix,`  
`std::size_t threads,`  
`const Checker & checker ) [inline]`

Constructor.

##### Parameters

|                         |                             |
|-------------------------|-----------------------------|
| <code>tempPrefix</code> | Prefix for temporary files. |
|-------------------------|-----------------------------|

**0.15.114.2.2 ~Consumer()** `delta::Consumer::~Consumer ( ) [inline]`  
Destructor.

#### 0.15.114.3 Member Function Documentation

**0.15.114.3.1 consume()** `void delta::Consumer::consume (`  
`const Node & n,`  
`const std::string & message,`  
`std::size_t num ) [inline]`

Initiate asynchronous check for the given node.

**Parameters**

|                |                                       |
|----------------|---------------------------------------|
| <i>n</i>       | <a href="#">Node</a> to check.        |
| <i>message</i> | Message.                              |
| <i>num</i>     | Number of this node in the iteration. |

**0.15.114.3.2 `getProgress()`** `std::pair<unsigned, unsigned> delta::Consumer::getProgress ( ) const [inline]`  
Returns the current progress.

**Returns**

Pair of finished and total checks.

**0.15.114.3.3 `getResult()`** `std::tuple<Node, std::string, std::size_t> delta::Consumer::getResult ( ) const [inline]`  
Returns the result, assuming that one exists.

**Returns**

Result.

**0.15.114.3.4 `hasResult()`** `bool delta::Consumer::hasResult ( ) const [inline]`  
Checks if at least one job was successful.

**Returns**

If a result is there.

**0.15.114.3.5 `reset()`** `void delta::Consumer::reset ( ) [inline]`  
Reset this executor.  
Waits for all jobs to finish and resets the internal status.

**0.15.114.3.6 `wait()`** `bool delta::Consumer::wait ( ) [inline]`  
Wait for at least one job to finish.

**Returns**

If all jobs have finished.

## 0.15.115 `smrat::LRAModule< Settings >::Context Struct Reference`

Stores a formula, being part of the received formula of this module, and the position of this formula in the passed formula.

```
#include <LRAModule.h>
```

### Public Member Functions

- `Context (const FormulaT &_origin, ModuleInput::iterator _pos)`

## Data Fields

- `FormulaT origin`  
*The formula in the received formula.*
- `ModuleInput::iterator position`  
*The position of this formula in the passed formula.*

### 0.15.115.1 Detailed Description

```
template<class Settings>
struct smrat::LRAbstractModule< Settings >::Context
```

Stores a formula, being part of the received formula of this module, and the position of this formula in the passed formula.

TODO: Maybe it is enough to store a mapping of the formula to the position.

### 0.15.115.2 Constructor & Destructor Documentation

```
0.15.115.2.1 Context() template<class Settings >
smrat::LRAbstractModule< Settings >::Context::Context (
 const FormulaT & _origin,
 ModuleInput::iterator _pos) [inline]
```

### 0.15.115.3 Field Documentation

```
0.15.115.3.1 origin template<class Settings >
FormulaT smrat::LRAbstractModule< Settings >::Context::origin
The formula in the received formula.
```

```
0.15.115.3.2 position template<class Settings >
ModuleInput::iterator smrat::LRAbstractModule< Settings >::Context::position
The position of this formula in the passed formula.
```

## 0.15.116 smrat::icp::ContractionCandidate Class Reference

```
#include <ContractionCandidate.h>
```

### Public Member Functions

- `ContractionCandidate (const ContractionCandidate &_original)=delete`  
*Constructors:*
  - `ContractionCandidate ()=delete`
  - `ContractionCandidate (carl::Variable _lhs, const Poly _rhs, const ConstraintT &_constraint, carl::Variable _← derivationVar, Contractor< carl::SimpleNewton > &_contractor, const FormulaT &_origin, unsigned _id, bool _usePropagation)`
  - `ContractionCandidate (carl::Variable _lhs, const Poly _rhs, const ConstraintT &_constraint, carl::Variable _← derivationVar, Contractor< carl::SimpleNewton > &_contractor, unsigned _id, bool _usePropagation)`  
*Constructor only for nonlinear candidates.*
- `~ContractionCandidate ()`  
*Destructor:*
  - `const Poly & rhs () const`
- `Functions:`

- const `ConstraintT & constraint () const`
- `Contractor< carl::SimpleNewton > & contractor () const`
- bool `contract (EvalDoubleIntervalMap &_intervals, DoubleInterval &_resA, DoubleInterval &_resB)`
- `carl::Variable::Arg derivationVar () const`
- const `Poly & derivative () const`
- `carl::Variable::Arg lhs () const`
- const `FormulaSetT & origin () const`
- `FormulaSetT & rOrigin ()`
- void `addOrigin (const FormulaT &_origin)`
- void `removeOrigin (const FormulaT &_origin)`
- bool `hasOrigin (const FormulaT &_origin) const`
- void `setLinear ()`
- void `setNonlinear ()`
- bool `is_linear () const`
- unsigned `reusagesAfterTargetDiameterReached () const`
- unsigned `incrementReusagesAfterTargetDiameterReached ()`
- void `resetReusagesAfterTargetDiameterReached ()`
- void `addICPVariable (IcpVariable *_icpVar)`
- double `calcRWA ()`
- double `RWA () const`
- double `lastRWA () const`
- void `updateLastRWA ()`
- double `lastPayoff () const`
- unsigned `id () const`
- void `setDerivationVar (carl::Variable _var)`
- void `setLhs (carl::Variable _lhs)`
- void `setPayoff (double _weight)`
- void `calcDerivative () throw ()`
- size\_t `activity ()`
- bool `isActive () const`
- bool `isDerived () const`
- void `resetWeights ()`
- void `print (std::ostream &_out=std::cout) const`
- bool `operator< (ContractionCandidate const &rhs) const`
- bool `operator==(ContractionCandidate const &rhs) const`

### 0.15.116.1 Constructor & Destructor Documentation

**0.15.116.1.1 ContractionCandidate() [1/4]** smtrat::icp::ContractionCandidate::ContractionCandidate  
(  
    const `ContractionCandidate & _original ) [delete]`

Constructors:

**0.15.116.1.2 ContractionCandidate() [2/4]** smtrat::icp::ContractionCandidate::ContractionCandidate  
( ) [delete]

**0.15.116.1.3 ContractionCandidate()** [3/4] smtrat::icp::ContractionCandidate::ContractionCandidate

```
(
 carl::Variable _lhs,
 const Poly _rhs,
 const ConstraintT & _constraint,
 carl::Variable _derivationVar,
 Contractor< carl::SimpleNewton > & _contractor,
 const FormulaT & _origin,
 unsigned _id,
 bool _usePropagation) [inline]
```

**0.15.116.1.4 ContractionCandidate()** [4/4] smtrat::icp::ContractionCandidate::ContractionCandidate

```
(
 carl::Variable _lhs,
 const Poly _rhs,
 const ConstraintT & _constraint,
 carl::Variable _derivationVar,
 Contractor< carl::SimpleNewton > & _contractor,
 unsigned _id,
 bool _usePropagation) [inline]
```

Constructor only for nonlinear candidates.

#### Parameters

|                             |  |
|-----------------------------|--|
| <code>_constraint</code>    |  |
| <code>_derivationVar</code> |  |

**0.15.116.1.5 ~ContractionCandidate()** smtrat::icp::ContractionCandidate::~ContractionCandidate (

) [inline]

Destructor:

**0.15.116.2 Member Function Documentation****0.15.116.2.1 activity()** size\_t smtrat::icp::ContractionCandidate::activity () [inline]**0.15.116.2.2 addICPVariable()** void smtrat::icp::ContractionCandidate::addICPVariable (  
 IcpVariable \* \_icpVar ) [inline]**0.15.116.2.3 addOrigin()** void smtrat::icp::ContractionCandidate::addOrigin (  
 const FormulaT & \_origin )**0.15.116.2.4 calcDerivative()** void smtrat::icp::ContractionCandidate::calcDerivative () throw (  
) [inline]**0.15.116.2.5 calcRWA()** double smtrat::icp::ContractionCandidate::calcRWA () [inline]

**0.15.116.2.6 constraint()** const `ConstraintT`& `smtrat::icp::ContractionCandidate::constraint` ( )  
const [inline]

**0.15.116.2.7 contract()** bool `smtrat::icp::ContractionCandidate::contract` (  
`EvalDoubleIntervalMap` & `_intervals`,  
`DoubleInterval` & `_resA`,  
`DoubleInterval` & `_resB`) [inline]

**0.15.116.2.8 contractor()** `Contractor<carl::SimpleNewton>&` `smtrat::icp::ContractionCandidate::contractor` ( ) const [inline]

**0.15.116.2.9 derivationVar()** `carl::Variable::Arg` `smtrat::icp::ContractionCandidate::derivationVar` ( ) const [inline]

**0.15.116.2.10 derivative()** const `Poly&` `smtrat::icp::ContractionCandidate::derivative` ( ) const [inline]

**0.15.116.2.11 hasOrigin()** bool `smtrat::icp::ContractionCandidate::hasOrigin` ( )  
const `FormulaT` & `_origin` ) const [inline]

**0.15.116.2.12 id()** unsigned `smtrat::icp::ContractionCandidate::id` ( ) const [inline]

**0.15.116.2.13 incrementReusagesAfterTargetDiameterReached()** unsigned `smtrat::icp::ContractionCandidate::incrementReusagesAfterTargetDiameterReached` ( ) [inline]

**0.15.116.2.14 is\_linear()** bool `smtrat::icp::ContractionCandidate::is_linear` ( ) const [inline]

**0.15.116.2.15 isActive()** bool `smtrat::icp::ContractionCandidate::isActive` ( ) const [inline]

**0.15.116.2.16 isDerived()** bool `smtrat::icp::ContractionCandidate::isDerived` ( ) const [inline]

**0.15.116.2.17 lastPayoff()** double `smtrat::icp::ContractionCandidate::lastPayoff` ( ) const [inline]

**0.15.116.2.18 lastRWA()** double `smtrat::icp::ContractionCandidate::lastRWA` ( ) const [inline]

**0.15.116.2.19 lhs()** `carl::Variable::Arg` `smtrat::icp::ContractionCandidate::lhs` ( ) const [inline]

**0.15.116.2.20 operator<()** bool `smtrat::icp::ContractionCandidate::operator<` (  
`ContractionCandidate` const & `rhs`) const [inline]

**0.15.116.2.21 operator==()** `bool smtrat::icp::ContractionCandidate::operator== ( ContractionCandidate const & rhs ) const [inline]`

**0.15.116.2.22 origin()** `const FormulaSetT& smtrat::icp::ContractionCandidate::origin ( ) const [inline]`

**0.15.116.2.23 print()** `void smtrat::icp::ContractionCandidate::print ( std::ostream & _out = std::cout ) const [inline]`

**0.15.116.2.24 removeOrigin()** `void smtrat::icp::ContractionCandidate::removeOrigin ( const FormulaT & _origin )`

**0.15.116.2.25 resetReusagesAfterTargetDiameterReached()** `void smtrat::icp::ContractionCandidate::resetReusagesAfterTargetDiameterReached ( ) [inline]`

**0.15.116.2.26 resetWeights()** `void smtrat::icp::ContractionCandidate::resetWeights ( ) [inline]`

**0.15.116.2.27 reusagesAfterTargetDiameterReached()** `unsigned smtrat::icp::ContractionCandidate::reusagesAfterTargetDiameterReached ( ) const [inline]`

**0.15.116.2.28 rhs()** `const Poly& smtrat::icp::ContractionCandidate::rhs ( ) const [inline]`  
Functions:

**0.15.116.2.29 rOrigin()** `FormulaSetT& smtrat::icp::ContractionCandidate::rOrigin ( ) [inline]`

**0.15.116.2.30 RWA()** `double smtrat::icp::ContractionCandidate::RWA ( ) const [inline]`

**0.15.116.2.31 setDerivationVar()** `void smtrat::icp::ContractionCandidate::setDerivationVar ( carl::Variable _var ) [inline]`

**0.15.116.2.32 setLhs()** `void smtrat::icp::ContractionCandidate::setLhs ( carl::Variable _lhs ) [inline]`

**0.15.116.2.33 setLinear()** `void smtrat::icp::ContractionCandidate::setLinear ( ) [inline]`

**0.15.116.2.34 setNonlinear()** `void smtrat::icp::ContractionCandidate::setNonlinear ( ) [inline]`

**0.15.116.2.35 setPayoff()** `void smtrat::icp::ContractionCandidate::setPayoff ( double _weight ) [inline]`

**0.15.116.2.36 updateLastRWA()** void smrat::icp::ContractionCandidate::updateLastRWA ( ) [inline]

## 0.15.117 smrat::icp::contractionCandidateComp Struct Reference

```
#include <ContractionCandidate.h>
```

### Public Member Functions

- bool `operator()` (const `ContractionCandidate` \*const lhs, const `ContractionCandidate` \*const rhs) const

### 0.15.117.1 Member Function Documentation

**0.15.117.1.1 operator()** bool smrat::icp::contractionCandidateComp::operator() ( const `ContractionCandidate` \*const lhs, const `ContractionCandidate` \*const rhs ) const [inline]

## 0.15.118 smrat::icp::ContractionCandidateManager Class Reference

```
#include <ContractionCandidateManager.h>
```

### Public Member Functions

- `ContractionCandidateManager()`  
*Constructors.*
- `~ContractionCandidateManager()`
- `ContractionCandidate * createCandidate (carl::Variable _lhs, const Poly _rhs, const ConstraintT &_constraint, carl::Variable _derivationVar, Contractor< carl::SimpleNewton > &_contractor, bool _usePropagation)`  
*Constructor & Functions.*
- unsigned `getId (const ContractionCandidate *const _candidate) const`  
*Returns the id of the given contraction candidate.*
- `ContractionCandidate * getCandidate (unsigned _id) const`  
*Returns the contraction candidate for the given id.*
- void `closure (const ContractionCandidate *const _candidate, std::set< const ContractionCandidate * > &_candidates) const`  
*Calculates the closure of a certain candidate according to the variables contained.*
- const std::vector< `ContractionCandidate *` > & `candidates ()`

### 0.15.118.1 Constructor & Destructor Documentation

**0.15.118.1.1 ContractionCandidateManager()** smrat::icp::ContractionCandidateManager::ContractionCandidateManager ( )  
Constructors.

**0.15.118.1.2 ~ContractionCandidateManager()** smrat::icp::ContractionCandidateManager::~ContractionCandidateManager ( ) [inline]

### 0.15.118.2 Member Function Documentation

**0.15.118.2.1 candidates()** const std::vector<[ContractionCandidate\\*](#)>& smtrat::icp::ContractionCandidateManager::candidates ( ) [inline]

**0.15.118.2.2 closure()** void smtrat::icp::ContractionCandidateManager::closure ( const [ContractionCandidate](#) \*const \_candidate, std::set< const [ContractionCandidate](#) \* > & \_candidates ) const

Calculates the closure of a certain candidate according to the variables contained.

#### Parameters

|                         |  |
|-------------------------|--|
| <code>_candidate</code> |  |
|-------------------------|--|

#### Returns

the set of candidates in the closure of `_candidate`

**0.15.118.2.3 createCandidate()** [ContractionCandidate](#) \* smtrat::icp::ContractionCandidateManager::createCandidate ( carl::Variable `_lhs`, const [Poly](#) `_rhs`, const [ConstraintT](#) & `_constraint`, carl::Variable `_derivationVar`, [Contractor](#)< carl::SimpleNewton > & `_contractor`, bool `_usePropagation` )

Constructor & Functions.

Creates a new candidate with an unique id.

#### Parameters

|                             |                                                                                        |
|-----------------------------|----------------------------------------------------------------------------------------|
| <code>_lhs</code>           | The slack/nonlinear variable which represents the constraint                           |
| <code>_constraint</code>    | The constraint which is to be replaced                                                 |
| <code>_derivationVar</code> | The variable from which the derivative is created when performing contraction          |
| <code>_origin</code>        | The pointer to the original formula, needed for assertions and removals of subformulas |

#### Returns

a pointer to the created contraction candidate

**0.15.118.2.4 getCandidate()** [ContractionCandidate](#) \* smtrat::icp::ContractionCandidateManager::getCandidate ( unsigned `_id` ) const

Returns the contraction candidate for the given id.

#### Parameters

|                  |  |
|------------------|--|
| <code>_id</code> |  |
|------------------|--|

**Returns**

the pointer to the contraction candidate

**0.15.118.2.5 `getId()`** `unsigned smtrat::icp::ContractionCandidateManager::getId (`  
`const ContractionCandidate *const _candidate ) const`

Returns the id of the given contraction candidate.

**Parameters**

|                         |  |
|-------------------------|--|
| <code>_candidate</code> |  |
|-------------------------|--|

**Returns**

id of the candidate

## 0.15.119 `smtrat::parser::conversion::Converter< To >` Struct Template Reference

#include <Conversions.h>

**Public Member Functions**

- template<typename From >  
  bool `operator()` (const From &, To &) const
- bool `operator()` (const To &from, To &to) const

### 0.15.119.1 Member Function Documentation

**0.15.119.1.1 `operator()` [1/2]** template<typename To >  
template<typename From >  
bool `smtrat::parser::conversion::Converter< To >::operator()` (  
  const From & ,  
  To & ) const [inline]

**0.15.119.1.2 `operator()` [2/2]** template<typename To >  
bool `smtrat::parser::conversion::Converter< To >::operator()` (  
  const To & *from*,  
  To & *to* ) const [inline]

## 0.15.120 `smtrat::parser::conversion::Converter< FormulaT >` Struct Reference

#include <Conversions.h>

**Public Member Functions**

- template<typename From >  
  bool `operator()` (const From &, `FormulaT` &) const
- bool `operator()` (const `FormulaT` &from, `FormulaT` &to) const
- bool `operator()` (const carl::Variable &from, `FormulaT` &to) const

### 0.15.120.1 Member Function Documentation

```
0.15.120.1.1 operator() [1/3] bool smtrat::parser::conversion::Converter< FormulaT >::operator()
(
 const carl::Variable & from,
 FormulaT & to) const [inline]
```

```
0.15.120.1.2 operator() [2/3] bool smtrat::parser::conversion::Converter< FormulaT >::operator()
(
 const FormulaT & from,
 FormulaT & to) const [inline]
```

```
0.15.120.1.3 operator() [3/3] template<typename From >
bool smtrat::parser::conversion::Converter< FormulaT >::operator() (
 const From & ,
 FormulaT &) const [inline]
```

## 0.15.121 smtrat::parser::conversion::Converter< Poly > Struct Reference

```
#include <Conversions.h>
```

### Public Member Functions

- template<typename From >  
  bool **operator()** (const From &, Poly &) const
- bool **operator()** (const Poly &from, Poly &to) const
- bool **operator()** (const carl::Variable &from, Poly &to) const
- bool **operator()** (const Rational &from, Poly &to) const

### 0.15.121.1 Member Function Documentation

```
0.15.121.1.1 operator() [1/4] bool smtrat::parser::conversion::Converter< Poly >::operator() (
 const carl::Variable & from,
 Poly & to) const [inline]
```

```
0.15.121.1.2 operator() [2/4] template<typename From >
bool smtrat::parser::conversion::Converter< Poly >::operator() (
 const From & ,
 Poly &) const [inline]
```

```
0.15.121.1.3 operator() [3/4] bool smtrat::parser::conversion::Converter< Poly >::operator() (
 const Poly & from,
 Poly & to) const [inline]
```

```
0.15.121.1.4 operator() [4/4] bool smtrat::parser::conversion::Converter< Poly >::operator() (
 const Rational & from,
 Poly & to) const [inline]
```

## 0.15.122 smtrat::parser::conversion::Converter< types::BVTerm > Struct Reference

```
#include <Conversions.h>
```

## Public Member Functions

- template<typename From >  
  bool **operator()** (const From &, types::BVTerm &) const
- bool **operator()** (const types::BVVariable &from, types::BVTerm &to) const
- bool **operator()** (const types::BVTerm &from, types::BVTerm &to) const
- bool **operator()** (const FixedWidthConstant< Integer > &from, types::BVTerm &to) const

### 0.15.122.1 Member Function Documentation

**0.15.122.1.1 operator() [1/4]** bool smtrat::parser::conversion::Converter< types::BVTerm >::operator()

```
 const FixedWidthConstant< Integer > & from,
 types::BVTerm & to) const [inline]
```

**0.15.122.1.2 operator() [2/4]** template<typename From >

```
bool smtrat::parser::conversion::Converter< types::BVTerm >::operator()
 const From & ,
 types::BVTerm &) const [inline]
```

**0.15.122.1.3 operator() [3/4]** bool smtrat::parser::conversion::Converter< types::BVTerm >::operator()

```
 const types::BVTerm & from,
 types::BVTerm & to) const [inline]
```

**0.15.122.1.4 operator() [4/4]** bool smtrat::parser::conversion::Converter< types::BVTerm >::operator()

```
 const types::BVVariable & from,
 types::BVTerm & to) const [inline]
```

## 0.15.123 benchmax::settings::CoreSettings Struct Reference

Core settings.

```
#include <Settings.h>
```

### Data Fields

- bool **show\_help**  
*Whether to show the command-line help.*
- bool **show\_settings**  
*Whether to show the configured settings.*
- bool **be\_verbose**  
*Whether to be more verbose.*
- bool **be\_quiet**  
*Whether to be more quiet.*
- std::string **config\_file**  
*Path of an optional config file.*
- long **start\_time** = std::time(nullptr)  
*Timestamp the binary was started.*

### 0.15.123.1 Detailed Description

Core settings.

### 0.15.123.2 Field Documentation

#### 0.15.123.2.1 **be\_quiet** bool benchmax::settings::CoreSettings::be\_quiet

Whether to be more quiet.

#### 0.15.123.2.2 **be\_verbose** bool benchmax::settings::CoreSettings::be\_verbose

Whether to be more verbose.

#### 0.15.123.2.3 **config\_file** std::string benchmax::settings::CoreSettings::config\_file

Path of an optional config file.

#### 0.15.123.2.4 **show\_help** bool benchmax::settings::CoreSettings::show\_help

Whether to show the command-line help.

#### 0.15.123.2.5 **show\_settings** bool benchmax::settings::CoreSettings::show\_settings

Whether to show the configured settings.

#### 0.15.123.2.6 **start\_time** long benchmax::settings::CoreSettings::start\_time = std::time(nullptr)

Timestamp the binary was started.

## 0.15.124 smtrat::settings::CoreSettings Struct Reference

```
#include <Settings.h>
```

### Data Fields

- bool [show\\_help](#)
- bool [show\\_info](#)
- bool [show\\_version](#)
- bool [show\\_settings](#)
- bool [show\\_cmake\\_options](#)
- bool [show\\_strategy](#)
- bool [show\\_license](#)
- std::string [config\\_file](#)

### 0.15.124.1 Field Documentation

#### 0.15.124.1.1 **config\_file** std::string smtrat::settings::CoreSettings::config\_file

#### 0.15.124.1.2 **show\_cmake\_options** bool smtrat::settings::CoreSettings::show\_cmake\_options

**0.15.124.1.3 show\_help** bool smtrat::settings::CoreSettings::show\_help

**0.15.124.1.4 show\_info** bool smtrat::settings::CoreSettings::show\_info

**0.15.124.1.5 show\_license** bool smtrat::settings::CoreSettings::show\_license

**0.15.124.1.6 show\_settings** bool smtrat::settings::CoreSettings::show\_settings

**0.15.124.1.7 show\_strategy** bool smtrat::settings::CoreSettings::show\_strategy

**0.15.124.1.8 show\_version** bool smtrat::settings::CoreSettings::show\_version

## 0.15.125 smtrat::parser::CoreTheory Struct Reference

Implements the core theory of the booleans.

```
#include <Core.h>
```

### Public Types

- using `OperatorType` = boost::variant< `carl::FormulaType` >

### Public Member Functions

- `CoreTheory (ParserState *state)`  
*Declare a new variable with the given name and the given sort.*
- `bool declareVariable (const std::string &name, const carl::Sort &sort, types::VariableType &result, TheoryError &errors)`  
*Resolve an if-then-else operator.*
- `bool handleDistinct (const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)`  
*Resolve a distinct operator.*
- `bool functionCall (const Identifier &identifier, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)`  
*Resolve another unknown function call.*
- `virtual bool resolveSymbol (const Identifier &, types::TermType &, TheoryError &errors)`  
*Resolve a symbol that was not declared within the ParserState.*
- template<typename T , typename Builder >  
  `FormulaT expandDistinct (const std::vector< T > &values, const Builder &neqBuilder)`
- `virtual bool instantiate (const types::VariableType &, const types::TermType &, types::TermType &, TheoryError &errors)`  
*Instantiate a variable within a term.*
- `virtual bool refreshVariable (const types::VariableType &, types::VariableType &, TheoryError &errors)`

### Static Public Member Functions

- `static void addSimpleSorts (qi::symbols< char, carl::Sort > &sorts)`
- `static void addConstants (qi::symbols< char, types::ConstType > &constants)`

**Data Fields**

- `ParserState * state`

**0.15.125.1 Detailed Description**

Implements the core theory of the booleans.

**0.15.125.2 Member Typedef Documentation**

**0.15.125.2.1 OperatorType** `using smtrat::parser::CoreTheory::OperatorType = boost::variant<carl::FormulaType>`

**0.15.125.3 Constructor & Destructor Documentation**

**0.15.125.3.1 CoreTheory()** `smtrat::parser::CoreTheory::CoreTheory ( ParserState * state )`

**0.15.125.4 Member Function Documentation**

**0.15.125.4.1 addConstants()** `void smtrat::parser::CoreTheory::addConstants ( qi::symbols< char, types::ConstType > & constants ) [static]`

**0.15.125.4.2 addSimpleSorts()** `void smtrat::parser::CoreTheory::addSimpleSorts ( qi::symbols< char, carl::Sort > & sorts ) [static]`

**0.15.125.4.3 declareVariable()** `bool smtrat::parser::CoreTheory::declareVariable ( const std::string &, const carl::Sort &, types::VariableType &, TheoryError & errors ) [virtual]`

Declare a new variable with the given name and the given sort.

Reimplemented from `smtrat::parser::AbstractTheory`.

**0.15.125.4.4 expandDistinct()** `template<typename T, typename Builder > FormulaT smtrat::parser::AbstractTheory::expandDistinct ( const std::vector< T > & values, const Builder & negBuilder ) [inline], [inherited]`

**0.15.125.4.5 functionCall()** `bool smtrat::parser::CoreTheory::functionCall ( const Identifier &, const std::vector< types::TermType > &, types::TermType &, TheoryError & errors ) [virtual]`

Resolve another unknown function call.

Reimplemented from `smtrat::parser::AbstractTheory`.

```
0.15.125.4.6 handleDistinct() bool smtrat::parser::CoreTheory::handleDistinct (
 const std::vector< types::TermType > & ,
 types::TermType & ,
 TheoryError & errors) [virtual]
```

Resolve a distinct operator.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.125.4.7 handleITE() bool smtrat::parser::CoreTheory::handleITE (
 const FormulaT & ,
 const types::TermType & ,
 const types::TermType & ,
 types::TermType & ,
 types::TermType & ,
 TheoryError & errors) [virtual]
```

Resolve an if-then-else operator.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.125.4.8 instantiate() virtual bool smtrat::parser::AbstractTheory::instantiate (
 const types::VariableType & ,
 const types::TermType & ,
 types::TermType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Instantiate a variable within a term.

Reimplemented in [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

```
0.15.125.4.9 refreshVariable() virtual bool smtrat::parser::AbstractTheory::refreshVariable (
 const types::VariableType & ,
 types::VariableType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Reimplemented in [smtrat::parser::BitvectorTheory](#).

```
0.15.125.4.10 resolveSymbol() virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
 const Identifier & ,
 types::TermType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Resolve a symbol that was not declared within the [ParserState](#).

This might be a symbol that actually uses indices, for example bitvector constants.

Reimplemented in [smtrat::parser::BitvectorTheory](#).

## 0.15.125.5 Field Documentation

```
0.15.125.5.1 state ParserState* smtrat::parser::AbstractTheory::state [inherited]
```

## 0.15.126 smtrat::parser::types::CoreTheory Struct Reference

Types of the core theory.

```
#include <TheoryTypes.h>
```

### Public Types

- `typedef mpl::vector< FormulaT, std::string > ConstTypes`
- `typedef mpl::vector< carl::Variable > VariableTypes`

- `typedef mpl::vector< FormulaT, std::string > ExpressionTypes`
- `typedef mpl::vector< FormulaT, std::string > TermTypes`
- `typedef carl::mpl_variant_of< TermTypes >::type TermType`

### 0.15.126.1 Detailed Description

Types of the core theory.

### 0.15.126.2 Member Typedef Documentation

#### 0.15.126.2.1 ConstTypes `typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::ConstT`

#### 0.15.126.2.2 ExpressionTypes `typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::Exp`

#### 0.15.126.2.3 TermType `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::CoreTheory::TermT`

#### 0.15.126.2.4 TermTypes `typedef mpl::vector<FormulaT, std::string> smtrat::parser::types::CoreTheory::TermT`

#### 0.15.126.2.5 VariableTypes `typedef mpl::vector<carl::Variable> smtrat::parser::types::CoreTheory::VariableType`

## 0.15.127 smtrat::cadcells::representation::covering< H > Struct Template Reference

```
#include <heuristics.h>
```

### Static Public Member Functions

- `template<typename T >`  
`static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &ders)`

### 0.15.127.1 Member Function Documentation

#### 0.15.127.1.1 compute() `template<CoveringHeuristic H>` `template<typename T >` `static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering< H >::compute (` `const std::vector< datastructures::SampledDerivationRef< T >> & ders ) [static]`

## 0.15.128 smtrat::mcsat::arithmetic::Covering Class Reference

Semantics: The space is divided into a number of intervals: (-oo,a][a,a](a,b)[b,b](b,oo) A bit is set if the constraints refutes the corresponding interval.

```
#include <Covering.h>
```

## Public Member Functions

- `Covering (std::size_t intervals)`
- `void add (const FormulaT &c, const carl::Bitset &b)`
- `bool conflicts () const`
- `std::size_t satisfyingInterval () const`
- `const auto & satisfyingSamples () const`
- `void buildConflictingCore (std::vector<FormulaT> &core) const`

## Friends

- `std::ostream & operator<< (std::ostream &os, const Covering &ri)`

### 0.15.128.1 Detailed Description

Semantics: The space is divided into a number of intervals: (-oo,a][a,a](a,b)[b,b](b,oo) A bit is set if the constraints refutes the corresponding interval.

### 0.15.128.2 Constructor & Destructor Documentation

#### 0.15.128.2.1 `Covering()` `smtrat::mcsat::arithmetic::Covering::Covering ( std::size_t intervals ) [inline]`

#### 0.15.128.3 Member Function Documentation

##### 0.15.128.3.1 `add()` `void smtrat::mcsat::arithmetic::Covering::add ( const FormulaT & c, const carl::Bitset & b ) [inline]`

##### 0.15.128.3.2 `buildConflictingCore()` `void smtrat::mcsat::arithmetic::Covering::buildConflictingCore ( std::vector<FormulaT> & core ) const [inline]`

##### 0.15.128.3.3 `conflicts()` `bool smtrat::mcsat::arithmetic::Covering::conflicts ( ) const [inline]`

##### 0.15.128.3.4 `satisfyingInterval()` `std::size_t smtrat::mcsat::arithmetic::Covering::satisfyingInterval ( ) const [inline]`

##### 0.15.128.3.5 `satisfyingSamples()` `const auto& smtrat::mcsat::arithmetic::Covering::satisfyingSamples ( ) const [inline]`

### 0.15.128.4 Friends And Related Function Documentation

#### 0.15.128.4.1 `operator<<` `std::ostream& operator<< ( std::ostream & os, const Covering & ri ) [friend]`

### 0.15.129 **smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST\_CELL\_COVERING >** Struct Reference

```
#include <heuristics_covering.h>
```

#### Static Public Member Functions

- template<typename T >  
static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &ders)

#### 0.15.129.1 Member Function Documentation

```
0.15.129.1.1 compute() template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
CoveringHeuristic::BIGGEST_CELL_COVERING >::compute (
 const std::vector< datastructures::SampledDerivationRef< T >> & ders) [inline],
[static]
```

### 0.15.130 **smtrat::cadcells::representation::covering< CoveringHeuristic::BIGGEST\_CELL\_COVERING\_EW >** Struct Reference

```
#include <heuristics_covering.h>
```

#### Static Public Member Functions

- template<typename T >  
static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &ders)

#### 0.15.130.1 Member Function Documentation

```
0.15.130.1.1 compute() template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
CoveringHeuristic::BIGGEST_CELL_COVERING_EW >::compute (
 const std::vector< datastructures::SampledDerivationRef< T >> & ders) [inline],
[static]
```

### 0.15.131 **smtrat::cadcells::representation::covering< CoveringHeuristic::CHAIN\_COVERING >** Struct Reference

```
#include <heuristics_covering.h>
```

#### Static Public Member Functions

- template<typename T >  
static std::optional< datastructures::CoveringRepresentation< T > > compute (const std::vector< datastructures::SampledDerivationRef< T >> &ders)

#### 0.15.131.1 Member Function Documentation

```
0.15.131.1.1 compute() template<typename T >
static std::optional<datastructures::CoveringRepresentation<T> > smtrat::cadcells::representation::covering<
CoveringHeuristic::CHAIN_COVERING >::compute (
 const std::vector< datastructures::SampledDerivationRef< T >> & ders) [inline],
[static]
```

## 0.15.132 smtrat::cadcells::datastructures::CoveringDescription Class Reference

Describes a covering of the real line by SymbolicIntervals (given an appropriate sample).

```
#include <roots.h>
```

### Public Member Functions

- void **add** (const [SymbolicInterval](#) &c)  
*Add a [SymbolicInterval](#) to the covering.*
- const auto & **cells** () const

### 0.15.132.1 Detailed Description

Describes a covering of the real line by SymbolicIntervals (given an appropriate sample).

### 0.15.132.2 Member Function Documentation

```
0.15.132.2.1 add() void smtrat::cadcells::datastructures::CoveringDescription::add (
 const SymbolicInterval & c) [inline]
```

Add a [SymbolicInterval](#) to the covering.

The added cell needs to be the rightmost cell of the already added cells and not be contained in any of these cells (or vice versa).

- The first cell needs to have -oo as left bound.
- The last cell needs to have oo as right bound.
- All cells need to cover the real line under an appropriate sample.
- Evaluated under an appropriate sample, the left bound of the added cell c is strictly greater than the left bounds of already added cells (considering also "strictness" of the bounds).
- The right bound of the added cell c needs to be greater than all right bounds of already added cells (considering also "strictness" of the bounds).

```
0.15.132.2.2 cells() const auto& smtrat::cadcells::datastructures::CoveringDescription::cells (
) const [inline]
```

## 0.15.133 smtrat::cadcells::datastructures::CoveringRepresentation< P > Struct Template Reference

Represents a covering over a cell.

```
#include <representation.h>
```

## Public Member Functions

- `CoveringDescription get_covering () const`  
*Returns a descriptions of the covering.*
- `std::vector< SampledDerivationRef< P > > sampled_derivations ()`  
*Returns the derivations.*
- `bool is_valid () const`  
*Checks whether this represents a proper non-redundant covering.*

## Data Fields

- `std::vector< CellRepresentation< P > > cells`  
*Cells of the covering in increasing order and no cell is contained in another cell.*
- `IndexedRootOrdering ordering`  
*An ordering on the roots for the cell boundaries mainting the covering.*

### 0.15.133.1 Detailed Description

```
template<typename P>
struct smtrat::cadcells::datastructures::CoveringRepresentation< P >
```

Represents a covering over a cell.

The cells forming the covering are in increasing order (ordered by lower bound) and no cell is contained in another cell.

### 0.15.133.2 Member Function Documentation

**0.15.133.2.1 `get_covering()`** template<typename P >  
`CoveringDescription smtrat::cadcells::datastructures::CoveringRepresentation< P >::get_←`  
`covering ( ) const [inline]`  
 Returns a descriptions of the covering.

**0.15.133.2.2 `is_valid()`** template<typename P >  
`bool smtrat::cadcells::datastructures::CoveringRepresentation< P >::is_valid ( ) const [inline]`  
 Checks whether this represents a proper non-redundant covering.

**0.15.133.2.3 `sampled_derivations()`** template<typename P >  
`std::vector<SampledDerivationRef<P> > smtrat::cadcells::datastructures::CoveringRepresentation< P >::sampled_derivations ( ) [inline]`  
 Returns the derivations.

### 0.15.133.3 Field Documentation

**0.15.133.3.1 `cells`** template<typename P >  
`std::vector<CellRepresentation<P> > smtrat::cadcells::datastructures::CoveringRepresentation< P >::cells`  
 Cells of the covering in increasing order and no cell is contained in another cell.

**0.15.133.3.2 `ordering`** template<typename P >  
`IndexedRootOrdering smtrat::cadcells::datastructures::CoveringRepresentation< P >::ordering`  
 An ordering on the roots for the cell boundaries mainting the covering.

## 0.15.134 smtrat::mcsat::onecellcad::recursive::CoverNullification Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr bool `cover_nullification` = true

#### 0.15.134.1 Field Documentation

**0.15.134.1.1 cover\_nullification** constexpr bool smtrat::mcsat::onecellcad::recursive::Cover↔  
Nullification::cover\_nullification = true [static], [constexpr]

## 0.15.135 smtrat::CSplitModule< Settings > Class Template Reference

```
#include <CSplitModule.h>
```

### Public Types

- typedef `Settings` `SettingsType`
- enum class `LemmaType` : unsigned { `NORMAL` = 0 , `PERMANENT` = 1 }

### Public Member Functions

- std::string `moduleName` () const
- `CSplitModule` (const `ModuleInput` \*`_formula`, `Conditionals` &`_conditionals`, `Manager` \*`_manager=nullptr`)
- bool `addCore` (`ModuleInput`::const\_iterator `_subformula`)  
*The module has to take the given sub-formula of the received formula into account.*
- void `removeCore` (`ModuleInput`::const\_iterator `_subformula`)  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- void `updateModel` () const  
*Updates the current assignment into the model.*
- `Answer` `checkCore` ()  
*Checks the received formula for consistency.*
- bool `inform` (const `FormulaT` &`_constraint`)  
*Informs the module about the given constraint.*
- void `deinform` (const `FormulaT` &`_constraint`)  
*The inverse of informing about a constraint.*
- virtual void `init` ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- bool `add` (`ModuleInput`::const\_iterator `_subformula`)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual `Answer` `check` (bool `_final=false`, bool `_full=true`, `carl::Variable` `_objective=carl::Variable::NO_VARIABLE`)  
*Checks the received formula for consistency.*
- virtual void `remove` (`ModuleInput`::const\_iterator `_subformula`)  
*Removes everything related to the given sub-formula of the received formula.*
- virtual void `updateAllModels` ()  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< `carl::Variable` > > `getModelEqualities` () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned `currentlySatisfiedByBackend` (const `FormulaT` &`_formula`) const
- virtual unsigned `currentlySatisfied` (const `FormulaT` &) const

- bool `receivedVariable` (carl::Variable::Arg \_var) const
- `Answer solverState ()` const
- std::size\_t `id ()` const
- void `setId (std::size_t _id)`

*Sets this module's unique ID to identify itself.*
- `thread_priority threadPriority ()` const
- void `setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput * pReceivedFormula ()` const
- const `ModuleInput & rReceivedFormula ()` const
- const `ModuleInput * pPassedFormula ()` const
- const `ModuleInput & rPassedFormula ()` const
- const `Model & model ()` const
- const std::vector< `Model` > & `allModels ()` const
- const std::vector< `FormulaSetT` > & `infeasibleSubsets ()` const
- const std::vector< `Module` \* > & `usedBackends ()` const
- const carl::FastSet< `FormulaT` > & `constraintsToInform ()` const
- const carl::FastSet< `FormulaT` > & `informedConstraints ()` const
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas ()` const
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula ()` const
- `ModuleInput::const_iterator firstSubformulaToPass ()` const
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smtsat::Conditionals & answerFound ()` const
- bool `isPreprocessor ()` const
- carl::Variable `objective ()` const
- bool `is_minimizing ()` const
- void `excludeNotReceivedVariablesFromModel ()` const

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins)` const

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins)` const
- bool `isValidInfeasibleSubset ()` const
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1)` const

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, `FormulaT` > `getReceivedFormulaSimplified ()`
- void `print (const std::string &_initiation="***")` const

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***")` const

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***")` const

- Prints the vector of passed formula.
  - void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const
    - Prints the infeasible subsets.
  - void `printModel` (std::ostream &\_out=std::cout) const
    - Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.
  - void `printAllModels` (std::ostream &\_out=std::cout)
    - Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

## Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5
  - The number of different variables to consider for a probable infinite loop of branchings.
- static std::vector<`Branching`> `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))
  - Stores the last branches in a cycle buffer.
- static size\_t `mFirstPosInLastBranches` = 0
  - The beginning of the cyclic buffer storing the last branches.
- static std::vector<`FormulaT`> `mOldSplittingVariables`
  - Reusable splitting variables.

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)
  - Informs the module about the given constraint.
- virtual void `deinformCore` (const `FormulaT` &\_constraint)
  - The inverse of informing about a constraint.
- bool `anAnswerFound` () const
  - Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void `clearModel` () const
  - Clears the assignment, if any was found.
- void `clearModels` () const
  - Clears all assignments, if any was found.
- void `cleanModel` () const
  - Substitutes variable occurrences by its model value in the model values of other variables.
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
  - Adds the given set of formulas in the received formula to the origins of the given passed formula.
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const
  - Gets the origins of the passed formula at the given position.
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair<`ModuleInput::iterator`, bool> `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair<`ModuleInput::iterator`, bool> `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr<std::vector<`FormulaT`>> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)

- **virtual void addConstraintToInform** (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- **std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula** (`ModuleInput::const_iterator` \_subformula)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- **std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula** (const `FormulaT` &\_formula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- **bool originInReceivedFormula** (const `FormulaT` &\_origin) const
 

*Adds the given formula to the passed formula with no origin.*
- **std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula** (const `FormulaT` &\_formula, const `std::shared_ptr< std::vector< FormulaT >>` &\_origins)
 

*Adds the given formula to the passed formula.*
- **std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula** (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- **void generateTrivialInfeasibleSubset** ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- **void receivedFormulasAsInfeasibleSubset** (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- **std::vector< FormulaT >::const\_iterator findBestOrigin** (const `std::vector< FormulaT >` &\_origins) const
 

*Get the infeasible subsets the given backend provides.*
- **void getInfeasibleSubsets** ()
 

*Copies the infeasible subsets of the passed formula.*
- **std::vector< FormulaSetT > getInfeasibleSubsets** (const `Module` &\_backend) const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- **const Model & backendsModel** () const
 

*Stores all models of a backend in the list of all models of this module.*
- **virtual Answer runBackends** (bool \_final, bool \_full, carl::Variable \_objective)
 

*Runs the backend solvers on the passed formula.*
- **virtual Answer runBackends** ()
 

*Runs the backend solvers on the passed formula.*
- **virtual ModuleInput::iterator eraseSubformulaFromPassedFormula** (`ModuleInput::iterator` \_subformula, bool \_ignoreOrigins=false)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- **void clearPassedFormula** ()
 

*Merges the two vectors of sets into the first one.*
- **size\_t determine\_smallest\_origin** (const `std::vector< FormulaT >` &origins) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- **bool probablyLooping** (const typename Poly::PolyType &\_branchingPolynomial, const `Rational` &\_branchingValue) const
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- **bool branchAt** (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, `std::vector< FormulaT >` &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- **bool branchAt** (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const `std::vector< FormulaT >` &&\_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- **bool branchAt** (carl::Variable::Arg \_var, const `Rational` &\_value, `std::vector< FormulaT >` &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*

- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())  
• void `splitUnequalConstraint` (const `FormulaT` &)  
*Adds the following lemmas for the given constraint p!=0.*  
• unsigned `checkModel` () const  
• void `addInformationRelevantFormula` (const `FormulaT` &formula)  
*Adds a formula to the InformationRelevantFormula.*  
• const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()  
*Gets all InformationRelevantFormulas.*  
• bool `isLemmaLevel` (`LemmaLevel` level)  
*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)  
*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`  
*A reference to the manager.*
- `Model` `mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`  
*Stores all satisfying assignments.*
- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals` `mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`  
*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator` `mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- carl::FastSet< `FormulaT` > `mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- carl::FastSet< `FormulaT` > `mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.135.1 Member Typedef Documentation

**0.15.135.1.1 SettingsType** `template<typename Settings >`  
`typedef smtrat::CSplitModule< Settings >::SettingsType`

### 0.15.135.2 Member Enumeration Documentation

**0.15.135.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.135.3 Constructor & Destructor Documentation

**0.15.135.3.1 CSplitModule()** `template<typename Settings >`  
`smtrat::CSplitModule< Settings >::CSplitModule (`  
 `const ModuleInput * _formula,`  
 `Conditionals & _conditionals,`  
 `Manager * _manager = nullptr )`

### 0.15.135.4 Member Function Documentation

**0.15.135.4.1 add()** `bool smtrat::Module::add (`  
 `ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.135.4.2 addConstraintToInform()** void smtrat::Module::addConstraintToInform ( const `FormulaT` & `_constraint` ) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.135.4.3 addCore()** template<typename Settings >

bool smtrat::CSplitModule< Settings >::addCore ( ModuleInput::const\_iterator `_subformula` ) [virtual]

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

False, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; True, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.135.4.4 addInformationRelevantFormula()** void smtrat::Module::addInformationRelevantFormula (

const `FormulaT` & `formula` ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.135.4.5 addLemma()** void smtrat::Module::addLemma (

const `FormulaT` & `_lemma`,

const `LemmaType` & `_lt` = `LemmaType::NORMAL`,

const `FormulaT` & `_preferredFormula` = `FormulaT( carl::FormulaType::TRUE )` ) [inline],

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.135.4.6 addOrigin()** `void smtrat::Module::addOrigin (`  
`ModuleInput::iterator _formula,`  
`const FormulaT & _origin ) [inline], [protected], [inherited]`

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.135.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator,bool>`  
`smtrat::Module::addReceivedSubformulaToPassedFormula (`  
`ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.135.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (`  
`const FormulaT & _formula ) [inline], [protected], [inherited]`

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.135.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (`  
`const FormulaT & _formula,`  
`const FormulaT & _origin ) [inline], [protected], [inherited]`

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.135.4.10 addSubformulaToPassedFormula() [3/3]** `std::pair<ModuleInput::iterator, bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.135.4.11 allModels()** `const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]`**Returns**

All satisfying assignments, if existent.

**0.15.135.4.12 anAnswerFound()** `bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.135.4.13 answerFound()** `const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]`**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.135.4.14 backendsModel()** `const Model & smrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.135.4.15** **branchAt()** [1/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.135.4.16** **branchAt()** [2/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.135.4.17** **branchAt()** [3/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.135.4.18** **branchAt()** [4/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.135.4.19 check() Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

```
0.15.135.4.20 checkCore() template<typename Settings >
Answer smtrat::CSplitModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; UNKNOWN, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.135.4.21 checkInfeasibleSubsetForMinimality() void smtrat::Module::checkInfeasibleSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.135.4.22 checkModel() unsigned smtrat::Module::checkModel () const [protected], [inherited]
```

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.135.4.23 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.135.4.24 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.135.4.25 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.135.4.26 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.135.4.27 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.135.4.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.135.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.135.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.135.4.31 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.135.4.32 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.135.4.33 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.135.4.34 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.135.4.35 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.135.4.36 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.135.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.135.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.135.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.135.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const` [inline], [inherited]

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.135.4.41 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const` [inline], [inherited]

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.135.4.42 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.135.4.43 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.135.4.44 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.135.4.45 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.135.4.46 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.135.4.47 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.135.4.48 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.135.4.49 getModelEqualities()** std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.135.4.50 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.135.4.51 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.135.4.52 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

#### Parameters

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.135.4.53 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.135.4.54 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.135.4.55 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

#### Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.135.4.56 `id()`** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.135.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.135.4.58 `inform()`** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before `assertSubformula` is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.135.4.59 `informBackends()`** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.135.4.60 `informCore()`** `virtual bool smtrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before `assertSubformula` is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.135.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.135.4.62 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.135.4.63 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.135.4.64 isLemmaLevel()** `bool smrat::Module::isLemmaLevel( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.135.4.65 isPreprocessor()** `bool smrat::Module::isPreprocessor( ) const [inline], [inherited]`

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.135.4.66 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.135.4.67 merge()** `std::vector< FormulaT > smrat::Module::merge( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.135.4.68 `model()`** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.135.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (`

```
const Model & _modelA,
const Model & _modelB) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.135.4.70 `moduleName()`** `template<typename Settings >`  
`std::string smtrat::CSplitModule< Settings >::moduleName ( ) const [inline], [virtual]`**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.135.4.71 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`**0.15.135.4.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`  
`const FormulaT & _origin ) const [protected], [inherited]`**0.15.135.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin (`  
`) [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.135.4.74 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.135.4.75 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.135.4.76 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.135.4.77 print()** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const [inherited]`  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.135.4.78 printAllModels()** `void smtrat::Module::printAllModels (`  
`std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.135.4.79 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets (`  
`const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.135.4.80 printModel()** `void smtrat::Module::printModel (`  
`std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

#### Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.135.4.81 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

#### Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.135.4.82 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

#### Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.135.4.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.135.4.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.135.4.85 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.135.4.86 receivedVariable()** `bool smrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.135.4.87 remove()** `void smrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

**0.15.135.4.88 removeCore()** `template<typename Settings > void smrat::CSplitModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula ) [virtual]`

Removes the subformula of the received formula at the given position to the considered ones of this module. Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

**Parameters**

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.135.4.89 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.135.4.90 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

**0.15.135.4.91 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const [inline], [inherited]`

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.135.4.92 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A reference to the formula passed to this module.

**0.15.135.4.93 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.135.4.94 runBackends() [2/2]** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.135.4.95 setId()** `void smtrat::Module::setId (`

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

**Parameters**

|                 |                                    |
|-----------------|------------------------------------|
| <code>id</code> | The id to set this module's id to. |
|-----------------|------------------------------------|

**0.15.135.4.96 setThreadPriority()** `void smtrat::Module::setThreadPriority (`

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.135.4.97 solverState()** `Answer smtrat::Module::solverState () const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.135.4.98 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.135.4.99 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.135.4.100 updateAllModels()** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.135.4.101 updateLemmas()** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.135.4.102 updateModel()** `template<typename Settings >`

`void smtrat::CSplitModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.135.4.103 usedBackends()** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.135.5 Field Documentation****0.15.135.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.135.5.2 mAllModels** `std::vector<Model>` `smtrat::Module::mAllModels` [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.135.5.3 mBackendsFoundAnswer** `std::atomic_bool*` `smtrat::Module::mBackendsFoundAnswer` [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.135.5.4 mConstraintsToInform** `carl::FastSet<FormulaT>` `smtrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.135.5.5 mFinalCheck** `bool` `smtrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.135.5.6 mFirstPosInLastBranches** `std::size_t` `smtrat::Module::mFirstPosInLastBranches` = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.135.5.7 mFirstSubformulaToPass** `ModuleInput::iterator` `smtrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.135.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator` `smtrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.135.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.135.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.135.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.135.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.135.5.13 mLastBranches** std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.135.5.14 mLemmas** std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.135.5.15 mModel** Model smtrat::Module::mModel [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.135.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.135.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.135.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.135.5.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.135.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.135.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.135.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.135.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.135.5.24 mUsedBackends** std::vector<Module\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.135.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

**0.15.136 smtrat::CubeLIAModule< Settings > Class Template Reference**

```
#include <CubeLIAModule.h>
```

**Public Types**

- **typedef Settings SettingsType**
- **enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }**

**Public Member Functions**

- **std::string moduleName () const**
- **CubeLIAModule (const ModuleInput \*\_formula, Conditionals &\_conditionals, Manager \*\_manager=nullptr)**
- **~CubeLIAModule ()**
- **bool addCore (ModuleInput::const\_iterator \_subformula)**

*The module has to take the given sub-formula of the received formula into account.*
- **void removeCore (ModuleInput::const\_iterator \_subformula)**

*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- **void updateModel () const**

*Updates the current assignment into the model.*
- **Answer checkCore ()**

*Checks the received formula for consistency.*
- **bool inform (const FormulaT &\_constraint)**

*Informs the module about the given constraint.*
- **void deinform (const FormulaT &\_constraint)**

*The inverse of informing about a constraint.*
- **virtual void init ()**

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- **bool add (ModuleInput::const\_iterator \_subformula)**

*The module has to take the given sub-formula of the received formula into account.*
- **virtual Answer check (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)**

*Checks the received formula for consistency.*
- **virtual void remove (ModuleInput::const\_iterator \_subformula)**

*Removes everything related to the given sub-formula of the received formula.*
- **virtual void updateAllModels ()**

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- **virtual std::list< std::vector< carl::Variable > > getModelEqualities () const**

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- **unsigned currentlySatisfiedByBackend (const FormulaT &\_formula) const**
- **virtual unsigned currentlySatisfied (const FormulaT &) const**
- **bool receivedVariable (carl::Variable::Arg \_var) const**
- **Answer solverState () const**
- **std::size\_t id () const**

- void `setId` (std::size\_t \_id)  
*Sets this module's unique ID to identify itself.*
- `thread_priority threadPriority () const`
- void `setThreadPriority (thread_priority _threadPriority)`  
*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput * pReceivedFormula () const`
- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const std::vector< `Model` > & `allModels () const`
- const std::vector< `FormulaSetT` > & `infeasibleSubsets () const`
- const std::vector< `Module` \* > & `usedBackends () const`
- const carl::FastSet< `FormulaT` > & `constraintsToInform () const`
- const carl::FastSet< `FormulaT` > & `informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`  
*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`  
*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`  
*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`  
*Notifies that the received formulas has been checked.*
- const `smtat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`  
*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `isValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, `FormulaT` > `getReceivedFormulaSimplified ()`
- void `print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*

- void `printModel` (std::ostream &\_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< `Branching` > `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< `FormulaT` > `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Removes the origin of the given formula from the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)  
*Removes all origins of the given formula from the passed formula.*
- void `informBackends` (const `FormulaT` &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*

- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const `Model` & `backendsModel` () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel` () const
- void `getBackendsAllModels` () const
 

*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends` (bool \_final, bool \_full, carl::Variable \_objective)
 

*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends` ()
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula` (`ModuleInput::iterator` \_subformula, bool \_ignoreOrigins=false)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula` ()
- std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const
 

*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &←\_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*

- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &`formula`)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
 

*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)
 

*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &`_modelA`, const `Model` &`_modelB`)
 

*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals `mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`

*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput`::iterator `mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.136.1 Member Typedef Documentation

**0.15.136.1.1 SettingsType** `template<typename Settings >`  
`typedef smtrat::CubeLIAModule< Settings >::SettingsType`

### 0.15.136.2 Member Enumeration Documentation

**0.15.136.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.136.3 Constructor & Destructor Documentation

**0.15.136.3.1 CubeLIAModule()** `template<typename Settings >`  
`smtrat::CubeLIAModule< Settings >::CubeLIAModule (`  
`const ModuleInput * _formula,`  
`Conditionals & _conditionals,`  
`Manager * _manager = nullptr )`

**0.15.136.3.2 ~CubeLIAModule()** `template<typename Settings >`  
`smtrat::CubeLIAModule< Settings >::~CubeLIAModule ( )`

### 0.15.136.4 Member Function Documentation

**0.15.136.4.1 add()** `bool smtrat::Module::add (`  
`ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.136.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`

```
const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.136.4.3 addCore()** `template<typename Settings >`

```
bool smtrat::CubeLIAModule< Settings >::addCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.136.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula`

```
(
```

```
const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.136.4.5 addLemma()** `void smtrat::Module::addLemma (`

```
const FormulaT & _lemma,
```

```
const LemmaType & _lt = LemmaType::NORMAL,
```

```
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
```

```
[inherited]
```

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.136.4.6 addOrigin()** `void smtrat::Module::addOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.136.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.136.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator, bool>` `smtrat::Module::addSubformulaToPassedFormula (`

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.136.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator, bool>` `smtrat::Module::addSubformulaToPassedFormula (`

```
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.136.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>**

```
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.136.4.11 allModels() const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]****Returns**

All satisfying assignments, if existent.

**0.15.136.4.12 anAnswerFound() bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.136.4.13 answerFound() const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.136.4.14 backendsModel() const Model & smtrat::Module::backendsModel () const [protected], [inherited]**

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.136.4.15** **branchAt()** [1/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.136.4.16** **branchAt()** [2/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.136.4.17** **branchAt()** [3/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.136.4.18** **branchAt()** [4/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.136.4.19 check() Answer smrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::PModule](#).

```
0.15.136.4.20 checkCore() template<typename Settings >
Answer smrat::CubeLIAModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

```
0.15.136.4.21 checkInfeasibleSubsetForMinimality() void smrat::Module::checkInfeasibleSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.136.4.22 checkModel() unsigned smrat::Module::checkModel () const [protected], [inherited]
```

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.136.4.23 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.136.4.24 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.136.4.25 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.136.4.26 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.136.4.27 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.136.4.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.136.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.136.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.136.4.31 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.136.4.32 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.136.4.33 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.136.4.34 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.136.4.35 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.136.4.36 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.136.4.37 eraseSubformulaFromPassedFormula()** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

```
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.136.4.38 excludeNotReceivedVariablesFromModel()** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.136.4.39 findBestOrigin()** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`

```
 const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

**Parameters**

|                       |
|-----------------------|
| <code>_origins</code> |
|-----------------------|

**Returns****0.15.136.4.40 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.136.4.41 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.136.4.42 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.136.4.43 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.136.4.44 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.136.4.45 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.136.4.46 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.136.4.47 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.136.4.48 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.136.4.49 getModelEqualities()** std::list< std::vector< `carl::Variable` > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.136.4.50 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.136.4.51 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.136.4.52 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

**Parameters**

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.136.4.53 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.136.4.54 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.136.4.55 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.136.4.56 `id()`** `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.136.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.136.4.58 `inform()`** `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.136.4.59 `informBackends()`** `void smtrat::Module::informBackends ( const FormulaT & _constraint ) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.136.4.60 `informCore()`** `virtual bool smtrat::Module::informCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.136.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.136.4.62 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.136.4.63 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.136.4.64 isLemmaLevel()** `bool smrat::Module::isLemmaLevel( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.136.4.65 isPreprocessor()** `bool smrat::Module::isPreprocessor( ) const [inline], [inherited]`

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.136.4.66 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.136.4.67 merge()** `std::vector< FormulaT > smrat::Module::merge( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.136.4.68 `model()`** `const Model& smtrat::Module::model( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.136.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.136.4.70 `moduleName()`** `template<typename Settings> std::string smtrat::CubeLIAModule< Settings >::moduleName( ) const [inline], [virtual]`**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.136.4.71 `objective()`** `carl::Variable smtrat::Module::objective( ) const [inline], [inherited]`**0.15.136.4.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula( const FormulaT & _origin ) const [protected], [inherited]`**0.15.136.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin( ) [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.136.4.74 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ()`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.136.4.75 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula () const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.136.4.76 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula () const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.136.4.77 print()** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const` [inherited]  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.136.4.78 printAllModels()** `void smtrat::Module::printAllModels (`  
`std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.136.4.79 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets (`  
`const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.136.4.80 printModel()** `void smtrat::Module::printModel (`  
`std::ostream & _out = std::cout ) const` [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.136.4.81 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.136.4.82 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.136.4.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.136.4.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.136.4.85 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.136.4.86 receivedVariable()** `bool smrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.136.4.87 remove()** `void smrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

**0.15.136.4.88 removeCore()** `template<typename Settings > void smrat::CubeLIAModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula ) [virtual]`

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.136.4.89 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.136.4.90 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

**0.15.136.4.91 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const [inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.136.4.92 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A reference to the formula passed to this module.

**0.15.136.4.93 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.136.4.94 runBackends() [2/2]** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.136.4.95 setId()** `void smtrat::Module::setId (`  
 `std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

**Parameters**

|                   |                                    |
|-------------------|------------------------------------|
| <code>↔ id</code> | The id to set this module's id to. |
|-------------------|------------------------------------|

**0.15.136.4.96 setThreadPriority()** `void smtrat::Module::setThreadPriority (`  
 `thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.136.4.97 solverState()** `Answer smtrat::Module::solverState () const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.136.4.98 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.136.4.99 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.136.4.100 updateAllModels()** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.136.4.101 updateLemmas()** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.136.4.102 updateModel()** `template<typename Settings >`

`void smtrat::CubeLIAModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.136.4.103 usedBackends()** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.136.5 Field Documentation****0.15.136.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.136.5.2 mAllModels** `std::vector<Model>` `smtrat::Module::mAllModels` [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.136.5.3 mBackendsFoundAnswer** `std::atomic_bool*` `smtrat::Module::mBackendsFoundAnswer` [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.136.5.4 mConstraintsToInform** `carl::FastSet<FormulaT>` `smtrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.136.5.5 mFinalCheck** `bool` `smtrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.136.5.6 mFirstPosInLastBranches** `std::size_t` `smtrat::Module::mFirstPosInLastBranches` = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.136.5.7 mFirstSubformulaToPass** `ModuleInput::iterator` `smtrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.136.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator` `smtrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.136.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.136.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.136.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.136.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.136.5.13 mLastBranches** std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.136.5.14 mLemmas** std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]

Stores the lemmas being valid formulas this module or its backends made.

**0.15.136.5.15 mModel** Model smtrat::Module::mModel [mutable], [protected], [inherited]

Stores the assignment of the current satisfiable result, if existent.

**0.15.136.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]

True, if the model has already been computed.

**0.15.136.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore =

5 [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.136.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]

Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.136.5.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]

Reusable splitting variables.

**0.15.136.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]

A reference to the manager.

**0.15.136.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]

Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.136.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.136.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.136.5.24 mUsedBackends** std::vector<Module\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.136.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

**0.15.137 smtrat::CurryModule< Settings > Class Template Reference**

```
#include <CurryModule.h>
```

**Public Types**

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

**Public Member Functions**

- `std::string moduleName () const`  
*Informs the module about the given constraint.*
- `CurryModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)`
- `~CurryModule ()`
- `bool informCore (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `bool addCore (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `void removeCore (ModuleInput::const_iterator _subformula)`  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- `void updateModel () const`  
*Updates the current assignment into the model.*
- `Answer checkCore ()`  
*Checks the received formula for consistency.*
- `bool inform (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void deform (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- `bool add (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
*Checks the received formula for consistency.*
- `virtual void remove (ModuleInput::const_iterator _subformula)`  
*Removes everything related to the given sub-formula of the received formula.*
- `virtual void updateAllModels ()`  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`

- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`

*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`
- `const std::vector< Model > & allModels () const`
- `const std::vector< FormulaSetT > & infeasibleSubsets () const`
- `const std::vector< Module * > & usedBackends () const`
- `const carl::FastSet< FormulaT > & constraintsToInform () const`
- `const carl::FastSet< FormulaT > & informedConstraints () const`
- `void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- `bool hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- `void clearLemmas ()`

*Deletes all yet found lemmas.*
- `const std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- `void receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- `const smtrat::Conditionals & answerFound () const`
- `bool isPreprocessor () const`
- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- `void updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- `virtual std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- `void printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- `void printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*

- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const  
*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector<`Branching`> `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector<`FormulaT`> `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual void `deinformCore` (const `FormulaT` &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair<`ModuleInput::iterator`, bool> `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair<`ModuleInput::iterator`, bool> `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr<std::vector<`FormulaT`>> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*

- std::pair< [ModuleInput::iterator](#), bool > [addReceivedSubformulaToPassedFormula](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool [originInReceivedFormula](#) (const [FormulaT](#) &\_origin) const
  - std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
  - std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
 

*Adds the given formula to the passed formula.*
  - std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
  - void [getInfeasibleSubsets](#) ()
 

*Copies the infeasible subsets of the passed formula.*
  - std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
  - const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - void [getBackendsModel](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
  - void [getBackendsAllModels](#) () const
 

*Runs the backend solvers on the passed formula.*
  - virtual [Answer](#) [runBackends](#) (bool \_final, bool \_full, carl::Variable \_objective)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
  - virtual [Answer](#) [runBackends](#) ()
 

*Merges the two vectors of sets into the first one.*
  - size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
  - bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
  - bool [branchAt](#) (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*branchAt (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())*
  - bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*branchAt (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())*
  - bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*branchAt (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())*

- void `splitUnequalConstraint` (const `FormulaT` &)  
*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &`formula`)  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()  
*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)  
*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &`_modelA`, const `Model` &`_modelB`)  
*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`  
*A reference to the manager.*
- `Model` `mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`  
*Stores all satisfying assignments.*
- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals `mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`  
*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput`::iterator `mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*

- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.137.1 Member Typedef Documentation

**0.15.137.1.1 SettingsType** `template<typename Settings >`  
`typedef smtrat::CurryModule< Settings >::SettingsType`

### 0.15.137.2 Member Enumeration Documentation

**0.15.137.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.137.3 Constructor & Destructor Documentation

**0.15.137.3.1 CurryModule()** `template<typename Settings >`  
`smtrat::CurryModule< Settings >::CurryModule (`  
 `const ModuleInput * _formula,`  
 `Conditionals & _conditionals,`  
 `Manager * _manager = nullptr )`

**0.15.137.3.2 ~CurryModule()** `template<typename Settings >`  
`smtrat::CurryModule< Settings >::~CurryModule ( )`

### 0.15.137.4 Member Function Documentation

**0.15.137.4.1 add()** `bool smtrat::Module::add (`  
 `ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.137.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`

`const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.137.4.3 addCore()** `template<typename Settings >`

`bool smtrat::CurryModule< Settings >::addCore (`  
`ModuleInput::const_iterator _subformula ) [virtual]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.137.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula (`

`const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.137.4.5 addLemma()** `void smtrat::Module::addLemma (`

`const FormulaT & _lemma,`

`const LemmaType & _lt = LemmaType::NORMAL,`

`const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],`

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

```
0.15.137.4.6 addOrigin() void smtrat::Module::addOrigin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

```
0.15.137.4.7 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

```
0.15.137.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat<-
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.137.4.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat<-
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.137.4.10 addSubformulaToPassedFormula() [3/3]** `std::pair<ModuleInput::iterator, bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.137.4.11 allModels()** `const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]`**Returns**

All satisfying assignments, if existent.

**0.15.137.4.12 anAnswerFound()** `bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.137.4.13 answerFound()** `const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]`**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.137.4.14 backendsModel()** `const Model & smrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.137.4.15** **branchAt()** [1/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.137.4.16** **branchAt()** [2/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.137.4.17** **branchAt()** [3/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.137.4.18** **branchAt()** [4/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

---

**0.15.137.4.19 check()** *Answer* smrat::Module::check (

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::PModule](#).

**0.15.137.4.20 checkCore()** template<typename Settings >  
*Answer* smrat::CurryModule< Settings >::checkCore ( ) [virtual]

Checks the received formula for consistency.

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.137.4.21 checkInfSubsetForMinimality()** void smrat::Module::checkInfSubsetForMinimality (

```
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.137.4.22 checkModel()** unsigned smrat::Module::checkModel ( ) const [protected], [inherited]

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.137.4.23 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.137.4.24 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.137.4.25 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.137.4.26 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.137.4.27 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.137.4.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.137.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.137.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.137.4.31 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.137.4.32 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.137.4.33 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.137.4.34 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.137.4.35 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.137.4.36 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.137.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.137.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.137.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.137.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.137.4.41 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.137.4.42 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.137.4.43 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.137.4.44 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.137.4.45 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.137.4.46 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.137.4.47 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.137.4.48 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.137.4.49 getModelEqualities()** std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.137.4.50 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.137.4.51 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.137.4.52 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

#### Parameters

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.137.4.53 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.137.4.54 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.137.4.55 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

#### Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.137.4.56 `id()`** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.137.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.137.4.58 `inform()`** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.137.4.59 `informBackends()`** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.137.4.60 `informCore()`** `template<typename Settings > bool smtrat::CurryModule< Settings >::informCore (const FormulaT & _constraint) [virtual]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.137.4.61 informedConstraints()** const carl::FastSet<[FormulaT](#)>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.137.4.62 init()** template<typename Settings >

void smtrat::CurryModule< Settings >::init ( ) [virtual]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not and will not be called more than once and only before the first runBackends call.

Reimplemented from [smtrat::Module](#).

**0.15.137.4.63 is\_minimizing()** bool smtrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.137.4.64 isLemmaLevel()** bool smtrat::Module::isLemmaLevel (

[LemmaLevel](#) level ) [protected], [inherited]

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.137.4.65 isPreprocessor()** bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.137.4.66 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.137.4.67 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge (

const std::vector< [FormulaT](#) > & \_vecSetA,

const std::vector< [FormulaT](#) > & \_vecSetB ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.137.4.68 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.137.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.137.4.70 `moduleName()`** `template<typename Settings> std::string smtrat::CurryModule<Settings>::moduleName () const [inline], [virtual]`**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.137.4.71 `objective()`** `carl::Variable smtrat::Module::objective () const [inline], [inherited]`**0.15.137.4.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (const FormulaT & _origin) const [protected], [inherited]`**0.15.137.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin () [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.137.4.74 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ()`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.137.4.75 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula () const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.137.4.76 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula () const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.137.4.77 print()** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const` [inherited]  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.137.4.78 printAllModels()** `void smtrat::Module::printAllModels (`  
`std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.137.4.79 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets (`  
`const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.137.4.80 printModel()** `void smtrat::Module::printModel (`  
`std::ostream & _out = std::cout ) const` [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.137.4.81 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.137.4.82 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.137.4.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.137.4.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.137.4.85 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.137.4.86 receivedVariable()** `bool smrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.137.4.87 remove()** `void smrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

**0.15.137.4.88 removeCore()** `template<typename Settings > void smrat::CurryModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula ) [virtual]`

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.137.4.89 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.137.4.90 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

**0.15.137.4.91 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const [inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.137.4.92 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A reference to the formula passed to this module.

**0.15.137.4.93 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.137.4.94 runBackends() [2/2]** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.137.4.95 setId()** `void smtrat::Module::setId (`

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

**Parameters**

|                   |                                    |
|-------------------|------------------------------------|
| <code>↔ id</code> | The id to set this module's id to. |
|-------------------|------------------------------------|

**0.15.137.4.96 setThreadPriority()** `void smtrat::Module::setThreadPriority (`

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.137.4.97 solverState()** `Answer smtrat::Module::solverState () const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.137.4.98 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.137.4.99 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.137.4.100 updateAllModels()** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.137.4.101 updateLemmas()** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.137.4.102 updateModel()** `template<typename Settings >`

`void smtrat::CurryModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.137.4.103 usedBackends()** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.137.5 Field Documentation****0.15.137.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.137.5.2 mAllModels** `std::vector<Model>` `smtrat::Module::mAllModels` [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.137.5.3 mBackendsFoundAnswer** `std::atomic_bool*` `smtrat::Module::mBackendsFoundAnswer` [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.137.5.4 mConstraintsToInform** `carl::FastSet<FormulaT>` `smtrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.137.5.5 mFinalCheck** `bool` `smtrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.137.5.6 mFirstPosInLastBranches** `std::size_t` `smtrat::Module::mFirstPosInLastBranches` = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.137.5.7 mFirstSubformulaToPass** `ModuleInput::iterator` `smtrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.137.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator` `smtrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.137.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.137.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.137.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.137.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.137.5.13 mLastBranches** std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.137.5.14 mLemmas** std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]

Stores the lemmas being valid formulas this module or its backends made.

**0.15.137.5.15 mModel** Model smtrat::Module::mModel [mutable], [protected], [inherited]

Stores the assignment of the current satisfiable result, if existent.

**0.15.137.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]

True, if the model has already been computed.

**0.15.137.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore =

5 [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.137.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]

Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.137.5.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]

Reusable splitting variables.

**0.15.137.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]

A reference to the manager.

**0.15.137.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]

Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.137.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.137.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.137.5.24 mUsedBackends** std::vector<[Module](#)\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.137.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.138 smtrat::CycleEnumerator< FHG, Collector > Struct Template Reference

This class encapsulates an algorithm for enumerating all cycles.

```
#include <ForwardHyperGraph.h>
```

### Public Member Functions

- [CycleEnumerator](#) (const FHG &fhg, Collector &c)

*Constructor from a graph and a collector object.*

- bool [findAll](#) ()

*Start the search for all cycles. Returns whether the search was finished (true) or aborted (false).*

### 0.15.138.1 Detailed Description

```
template<typename FHG, typename Collector>
struct smtrat::CycleEnumerator< FHG, Collector >
```

This class encapsulates an algorithm for enumerating all cycles.

This class implements an adaption of the algorithm described in [\[Johnson75\]](#). Unlike the algorithm described there, this version works on forward hypergraphs as implemented by [ForwardHyperGraph](#).

Although the algorithm works recursively, this implementation can be used somewhat iteratively. The cycles are given to the caller using an collector functor. If this functor returns true, the search for more cycles is aborted immediately. Due to this pattern, there is no need to store the cycles within this class and thus the memory usage is minimized.

### 0.15.138.2 Constructor & Destructor Documentation

#### 0.15.138.2.1 CycleEnumerator()

```
template<typename FHG , typename Collector >
smtrat::CycleEnumerator< FHG, Collector >::CycleEnumerator (
 const FHG & fhg,
 Collector & c) [inline]
```

Constructor from a graph and a collector object.

### 0.15.138.3 Member Function Documentation

#### 0.15.138.3.1 findAll()

```
template<typename FHG , typename Collector >
bool smtrat::CycleEnumerator< FHG, Collector >::findAll ()
```

Start the search for all cycles. Returns whether the search was finished (true) or aborted (false).

## 0.15.139 benchmax::Database Class Reference

Dummy database that effectively disables storing to database. Set BENCHMAX\_DATABASE to actually use a database.

```
#include <Database.h>
```

## Public Types

- using `Index` = `std::size_t`

*Dummy index type.*

## Public Member Functions

- `Index addTool (const Tool *)`  
*Dummy.*
- `Index getToolID (const Tool *)`  
*Dummy.*
- `Index addFile (const fs::path &)`  
*Dummy.*
- `Index getFileID (const fs::path &)`  
*Dummy.*
- `Index createBenchmark ()`  
*Dummy.*
- `Index addBenchmarkResult (Index, Index, Index, int, std::size_t)`  
*Dummy.*
- void `addBenchmarkAttribute (Index, const std::string &, const std::string &)`  
*Dummy.*

### 0.15.139.1 Detailed Description

Dummy database that effectively disables storing to database. Set BENCHMAX\_DATABASE to actually use a database.

### 0.15.139.2 Member Typedef Documentation

#### 0.15.139.2.1 `Index` using `benchmax::Database::Index` = `std::size_t`

Dummy index type.

### 0.15.139.3 Member Function Documentation

#### 0.15.139.3.1 `addBenchmarkAttribute()`

```
void benchmax::Database::addBenchmarkAttribute (
 Index ,
 const std::string & ,
 const std::string &) [inline]
```

Dummy.

#### 0.15.139.3.2 `addBenchmarkResult()`

```
Index benchmax::Database::addBenchmarkResult (
 Index ,
 Index ,
 Index ,
 int ,
 std::size_t) [inline]
```

Dummy.

**0.15.139.3.3 addFile()** `Index` `benchmax::Database::addFile (`  
    `const fs::path & )` [inline]  
Dummy.

**0.15.139.3.4 addTool()** `Index` `benchmax::Database::addTool (`  
    `const Tool * )` [inline]  
Dummy.

**0.15.139.3.5 createBenchmark()** `Index` `benchmax::Database::createBenchmark ( )` [inline]  
Dummy.

**0.15.139.3.6 getFileID()** `Index` `benchmax::Database::getFileID (`  
    `const fs::path & )` [inline]  
Dummy.

**0.15.139.3.7 getToolID()** `Index` `benchmax::Database::getToolID (`  
    `const Tool * )` [inline]  
Dummy.

## 0.15.140 benchmax::DBAL Class Reference

```
#include <Database_mysqlconnector.h>
```

### Public Types

- using `Index` = `std::size_t`  
*Indec type.*
- using `Statement` = `std::unique_ptr<sql::PreparedStatement>`  
*Statement type.*
- using `Results` = `std::unique_ptr<sql::ResultSet>`  
*Results type.*
- `typedef mysqlpp::sql_bigint_unsigned Index`
- `typedef mysqlpp::Query Statement`
- `typedef mysqlpp::StoreQueryResult Results`

### Public Member Functions

- `bool connect (const std::string &db, const std::string &host, const std::string &user, const std::string &password)`  
*Connect to the given database.*
- `Statement prepare (const std::string &query)`  
*Prepare a statement.*
- `Results execute (Statement &stmt)`  
*Execute a statement without arguments.*
- `Results execute (Statement &stmt, const std::string &a1)`  
*Execute a statement with a single string.*
- `Results execute (Statement &stmt, const std::string &a1, std::size_t a2)`  
*Execute a statement with a string and a number.*
- `Results execute (Statement &stmt, const std::string &a1, const std::string &a2, const Index &a3)`  
*Execute a statement with two strings and an index.*

- **Results execute** (`Statement &stmt, int a1, std::size_t a2, std::size_t a3, const Index &a4, const Index &a5, const Index &a6)`  
*Execute a statement with an int, two numbers and three indices.*
- template<typename... Args>  
**Index insert** (`const std::string &query, const Args &... args)`  
*Insert some data and return the new index.*
- template<typename... Args>  
**Results select** (`const std::string &query, const Args &... args)`  
*Select some data.*
- `std::size_t size` (`const Results &res)`  
*Return the size of the results.*
- **Index getIndex** (`const Results &res, const std::string &name)`  
*Get the index of some name within the results.*
- `bool connect` (`const std::string &db, const std::string &host, const std::string &user, const std::string &password)`
- **Statement prepare** (`const std::string &query)`
- template<typename... Args>  
`void execute` (`Statement &stmt, const Args &... args)`
- template<typename... Args>  
**Index insert** (`const std::string &query, const Args &... args)`
- template<typename... Args>  
**Results select** (`const std::string &query, const Args &... args)`
- `std::size_t size` (`const Results &res)`
- **Index getIndex** (`const Results &res, const std::string &name)`

#### 0.15.140.1 Member Typedef Documentation

**0.15.140.1.1 Index [1/2]** using `benchmax::DBAL::Index` = `std::size_t`  
Indec type.

**0.15.140.1.2 Index [2/2]** typedef `mysqlpp::sql bigint_unsigned` `benchmax::DBAL::Index`

**0.15.140.1.3 Results [1/2]** using `benchmax::DBAL::Results` = `std::unique_ptr<sql::ResultSet>`  
Results type.

**0.15.140.1.4 Results [2/2]** typedef `mysqlpp::StoreQueryResult` `benchmax::DBAL::Results`

**0.15.140.1.5 Statement [1/2]** using `benchmax::DBAL::Statement` = `std::unique_ptr<sql::PreparedStatement>`  
Statement type.

**0.15.140.1.6 Statement [2/2]** typedef `mysqlpp::Query` `benchmax::DBAL::Statement`

#### 0.15.140.2 Member Function Documentation

```
0.15.140.2.1 connect() [1/2] bool benchmax::DBAL::connect (
 const std::string & db,
 const std::string & host,
 const std::string & user,
 const std::string & password) [inline]
```

Connect to the given database.

```
0.15.140.2.2 connect() [2/2] bool benchmax::DBAL::connect (
 const std::string & db,
 const std::string & host,
 const std::string & user,
 const std::string & password) [inline]
```

```
0.15.140.2.3 execute() [1/6] Results benchmax::DBAL::execute (
 Statement & stmt) [inline]
```

Execute a statement without arguments.

```
0.15.140.2.4 execute() [2/6] template<typename... Args>
void benchmax::DBAL::execute (
 Statement & stmt,
 const Args &... args) [inline]
```

```
0.15.140.2.5 execute() [3/6] Results benchmax::DBAL::execute (
 Statement & stmt,
 const std::string & a1) [inline]
```

Execute a statement with a single string.

```
0.15.140.2.6 execute() [4/6] Results benchmax::DBAL::execute (
 Statement & stmt,
 const std::string & a1,
 const std::string & a2,
 const Index & a3) [inline]
```

Execute a statement with two strings and an index.

```
0.15.140.2.7 execute() [5/6] Results benchmax::DBAL::execute (
 Statement & stmt,
 const std::string & a1,
 std::size_t a2) [inline]
```

Execute a statement with a string and a number.

```
0.15.140.2.8 execute() [6/6] Results benchmax::DBAL::execute (
 Statement & stmt,
 int a1,
 std::size_t a2,
 std::size_t a3,
 const Index & a4,
 const Index & a5,
 const Index & a6) [inline]
```

Execute a statement with an int, two numbers and three indices.

**0.15.140.2.9 getIndex() [1/2]** `Index` `benchmax::DBAL::getIndex (`  
    `const Results & res,`  
    `const std::string & name )` [inline]

Get the index of some name within the results.

**0.15.140.2.10 getIndex() [2/2]** `Index` `benchmax::DBAL::getIndex (`  
    `const Results & res,`  
    `const std::string & name )` [inline]

**0.15.140.2.11 insert() [1/2]** `template<typename... Args>`  
`Index` `benchmax::DBAL::insert (`  
    `const std::string & query,`  
    `const Args &... args )` [inline]

Insert some data and return the new index.

**0.15.140.2.12 insert() [2/2]** `template<typename... Args>`  
`Index` `benchmax::DBAL::insert (`  
    `const std::string & query,`  
    `const Args &... args )` [inline]

**0.15.140.2.13 prepare() [1/2]** `Statement` `benchmax::DBAL::prepare (`  
    `const std::string & query )` [inline]

Prepare a statement.

**0.15.140.2.14 prepare() [2/2]** `Statement` `benchmax::DBAL::prepare (`  
    `const std::string & query )` [inline]

**0.15.140.2.15 select() [1/2]** `template<typename... Args>`  
`Results` `benchmax::DBAL::select (`  
    `const std::string & query,`  
    `const Args &... args )` [inline]

Select some data.

**0.15.140.2.16 select() [2/2]** `template<typename... Args>`  
`Results` `benchmax::DBAL::select (`  
    `const std::string & query,`  
    `const Args &... args )` [inline]

**0.15.140.2.17 size() [1/2]** `std::size_t` `benchmax::DBAL::size (`  
    `const Results & res )` [inline]

Return the size of the results.

**0.15.140.2.18 size() [2/2]** `std::size_t` `benchmax::DBAL::size (`  
    `const Results & res )` [inline]

### 0.15.141 smtrat::parser::DecimalParser Struct Reference

Parses decimals: numeral.0\*numeral  
#include <Lexicon.h>

#### 0.15.141.1 Detailed Description

Parses decimals: numeral.0\*numeral

### 0.15.142 Minisat::DeepEqual< K > Struct Template Reference

#include <Map.h>

#### Public Member Functions

- bool `operator()` (const K \*k1, const K \*k2) const

#### 0.15.142.1 Member Function Documentation

```
0.15.142.1.1 operator() template<class K >
bool Minisat::DeepEqual< K >::operator() (
 const K * k1,
 const K * k2) const [inline]
```

### 0.15.143 Minisat::DeepHash< K > Struct Template Reference

#include <Map.h>

#### Public Member Functions

- uint32\_t `operator()` (const K \*k) const

#### 0.15.143.1 Member Function Documentation

```
0.15.143.1.1 operator() template<class K >
uint32_t Minisat::DeepHash< K >::operator() (
 const K * k) const [inline]
```

### 0.15.144 smtrat::mcsat::fm::DefaultComparator Struct Reference

Does not order anything.

#include <ConflictGenerator.h>

#### Public Member Functions

- bool `operator()` (const Bound &, const Bound &) const

#### Data Fields

- bool `symmetric` = false

#### 0.15.144.1 Detailed Description

Does not order anything.

### 0.15.144.2 Member Function Documentation

```
0.15.144.2.1 operator() bool smtrat::mcsat::fm::DefaultComparator::operator() (const Bound &, const Bound &) const [inline]
```

### 0.15.144.3 Field Documentation

```
0.15.144.3.1 symmetric bool smtrat::mcsat::fm::DefaultComparator::symmetric = false
```

## 0.15.145 smtrat::mcsat::fm::DefaultSettings Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr bool `use_all_constraints` = false

### 0.15.145.1 Field Documentation

```
0.15.145.1.1 use_all_constraints constexpr bool smtrat::mcsat::fm::DefaultSettings::use_all_constraints = false [static], [constexpr]
```

## 0.15.146 smtrat::mcsat::smtaf::DefaultSettings Struct Reference

```
#include <AssignmentFinder.h>
```

### Static Public Attributes

- static constexpr unsigned int `assignAllVariables` = true
- static constexpr bool `advance_level_by_level` = false

*If set to true, a conflict on the lowest possible level is returned.*

### 0.15.146.1 Field Documentation

```
0.15.146.1.1 advance_level_by_level constexpr bool smtrat::mcsat::smtaf::DefaultSettings::advance_level_by_level = false [static], [constexpr]
```

If set to true, a conflict on the lowest possible level is returned.

Got more or less irrelevant as unassigned variables are not ordered anymore.

```
0.15.146.1.2 assignAllVariables constexpr unsigned int smtrat::mcsat::smtaf::DefaultSettings::assignAllVariables = true [static], [constexpr]
```

## 0.15.147 smtrat::mcsat::vs::DefaultSettings Struct Reference

```
#include <ExplanationGenerator.h>
```

**Static Public Attributes**

- static const bool `reduceConflictConstraints` = true
- static const bool `clauseChainWithEquivalences` = false

**0.15.147.1 Field Documentation**

**0.15.147.1.1 clauseChainWithEquivalences** const bool smtrat::mcsat::vs::DefaultSettings::clauseChainWithEquivalences = false [static]

**0.15.147.1.2 reduceConflictConstraints** const bool smtrat::mcsat::vs::DefaultSettings::reduceConflictConstraints = true [static]

**0.15.148 smtrat::cad::projection\_compare::degree Struct Reference**

```
#include <ProjectionComparator.h>
```

**0.15.149 smtrat::mcsat::onecellcad::recursive::DegreeAscending Struct Reference**

```
#include <Explanation.h>
```

**Static Public Attributes**

- static constexpr int `heuristic` = 1

**0.15.149.1 Field Documentation**

**0.15.149.1.1 heuristic** constexpr int smtrat::mcsat::onecellcad::recursive::DegreeAscending::heuristic = 1 [static], [constexpr]

**0.15.150 smtrat::analyzer::DegreeCollector Struct Reference**

```
#include <utils.h>
```

**Public Member Functions**

- void `operator()` (const `Poly` &p)
- void `operator()` (const carl::UnivariatePolynomial<`Poly`> &p)
- void `operator()` (const `ConstraintT` &c)

**Data Fields**

- std::size\_t `degree_max` = 0
- std::size\_t `degree_sum` = 0
- std::size\_t `tdegree_max` = 0
- std::size\_t `tdegree_sum` = 0

**0.15.150.1 Member Function Documentation**

**0.15.150.1.1 operator()() [1/3]** void smtrat::analyzer::DegreeCollector::operator() (const carl::UnivariatePolynomial<`Poly`> &p) [inline]

```
0.15.150.1.2 operator() [2/3] void smtrat::analyzer::DegreeCollector::operator() (
 const ConstraintT & c) [inline]
```

```
0.15.150.1.3 operator() [3/3] void smtrat::analyzer::DegreeCollector::operator() (
 const Poly & p) [inline]
```

## 0.15.150.2 Field Documentation

```
0.15.150.2.1 degree_max std::size_t smtrat::analyzer::DegreeCollector::degree_max = 0
```

```
0.15.150.2.2 degree_sum std::size_t smtrat::analyzer::DegreeCollector::degree_sum = 0
```

```
0.15.150.2.3 tdegree_max std::size_t smtrat::analyzer::DegreeCollector::tdegree_max = 0
```

```
0.15.150.2.4 tdegree_sum std::size_t smtrat::analyzer::DegreeCollector::tdegree_sum = 0
```

## 0.15.151 smtrat::mcsat::onecellcad::recursive::DegreeDescending Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr int heuristic = 2

## 0.15.151.1 Field Documentation

```
0.15.151.1.1 heuristic constexpr int smtrat::mcsat::onecellcad::recursive::DegreeDescending::
::heuristic = 2 [static], [constexpr]
```

## 0.15.152 smtrat::cadcells::datastructures::DelineatedDerivation< Properties > Class Template Reference

A DelineatedDerivation is a BaseDerivation with a Delineation and an underlying SampledDerivation.

```
#include <derivation.h>
```

### Public Member Functions

- DelineatedDerivation (BaseDerivationRef< Properties > base)
- DelineatedDerivation (BaseDerivationRef< Properties > base, const Delineation & delineation)
- BaseDerivationRef< Properties > & base ()
- const BaseDerivationRef< Properties > & base () const
- Delineation & delin ()
- const Assignment & underlying\_sample () const
- DerivationRef< Properties > & underlying ()
- const DerivationRef< Properties > & underlying () const
- PolyPool & polys ()
- Projections & proj ()
- carl::Variable main\_var () const

- `size_t level () const`
- `template<typename P >`  
`void insert (P property)`
- `template<typename P >`  
`bool contains (const P &property) const`
- `template<typename P >`  
`const PropertiesTSet< P > & properties () const`
- `void merge_with (const DelineatedDerivation< Properties > &other)`

### 0.15.152.1 Detailed Description

```
template<typename Properties>
class smtrat::cadcells::datastructures::DelineatedDerivation< Properties >
```

A `DelineatedDerivation` is a `BaseDerivation` with a `Delineation` and an underlying `SampledDerivation`.

#### Template Parameters

|                         |                   |
|-------------------------|-------------------|
| <code>Properties</code> | Set of properties |
|-------------------------|-------------------|

### 0.15.152.2 Constructor & Destructor Documentation

**0.15.152.2.1 `DelineatedDerivation()` [1/2]** `template<typename Properties >`  
`smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::DelineatedDerivation (`  
`BaseDerivationRef< Properties > base ) [inline]`

**0.15.152.2.2 `DelineatedDerivation()` [2/2]** `template<typename Properties >`  
`smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::DelineatedDerivation (`  
`BaseDerivationRef< Properties > base,`  
`const Delineation & delineation ) [inline]`

### 0.15.152.3 Member Function Documentation

**0.15.152.3.1 `base()` [1/2]** `template<typename Properties >`  
`BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::base ( ) [inline]`

**0.15.152.3.2 `base()` [2/2]** `template<typename Properties >`  
`const BaseDerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::base ( ) const [inline]`

**0.15.152.3.3 `contains()`** `template<typename Properties >`  
`template<typename P >`  
`bool smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::contains (`  
`const P & property ) const [inline]`

**0.15.152.3.4 `delin()`** template<typename Properties >  
Delineation& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::delin ( )  
[inline]

**0.15.152.3.5 `insert()`** template<typename Properties >  
template<typename P >  
void smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::insert ( P property ) [inline]

**0.15.152.3.6 `level()`** template<typename Properties >  
size\_t smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::level ( ) const  
[inline]

**0.15.152.3.7 `main_var()`** template<typename Properties >  
carl::Variable smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::main\_var ( ) const [inline]

**0.15.152.3.8 `merge_with()`** template<typename Properties >  
void smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::merge\_with ( const DelineatedDerivation< Properties > & other ) [inline]

**0.15.152.3.9 `polys()`** template<typename Properties >  
PolyPool& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::polys ( )  
[inline]

**0.15.152.3.10 `proj()`** template<typename Properties >  
Projections& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::proj ( )  
[inline]

**0.15.152.3.11 `properties()`** template<typename Properties >  
template<typename P >  
const PropertiesTSet<P>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::properties ( ) const [inline]

**0.15.152.3.12 `underlying()` [1/2]** template<typename Properties >  
DerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::underlying ( ) [inline]

**0.15.152.3.13 `underlying()` [2/2]** template<typename Properties >  
const DerivationRef<Properties>& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::underlying ( ) const [inline]

**0.15.152.3.14 `underlying_sample()`** template<typename Properties >  
const Assignment& smtrat::cadcells::datastructures::DelineatedDerivation< Properties >::underlying< \_sample ( ) const [inline]

## 0.15.153 smtrat::cadcells::datastructures::Delineation Class Reference

Represents the delineation of a set of polynomials (at a sample), that is.

```
#include <delineation.h>
```

### Public Member Functions

- [Delineation \(\)](#)  
• const auto & [roots \(\) const](#)  
*Returns the underlying root map, which is a set of real algebraic numbers to indexed root expressions.*
- const auto & [nullified \(\) const](#)  
*The set of nullified polynomials.*
- const auto & [nonzero \(\) const](#)  
*The set of polynomials without roots.*
- bool [empty \(\) const](#)
- auto [delineate\\_cell \(const RAN &sample\) const](#)  
*Computes the bounds of the interval around the given sample w.r.t.*
- void [add\\_root \(RAN root, TaggedIndexedRoot tagged\\_root\)](#)
- void [add\\_poly\\_nonzero \(PolyRef poly\)](#)
- void [add\\_poly\\_nullified \(PolyRef poly\)](#)

### Friends

- class [DelineationInterval](#)

### 0.15.153.1 Detailed Description

Represents the delineation of a set of polynomials (at a sample), that is.

- the ordering of all roots,
- the set of nullified polynomials,
- the set of polynomials without a root.

### 0.15.153.2 Constructor & Destructor Documentation

#### 0.15.153.2.1 Delineation() `smtrat::cadcells::datastructures::Delineation () [inline]`

### 0.15.153.3 Member Function Documentation

#### 0.15.153.3.1 add\_poly\_nonzero() `void smtrat::cadcells::datastructures::Delineation::add_poly_← nonzero ( PolyRef poly ) [inline]`

#### 0.15.153.3.2 add\_poly\_nullified() `void smtrat::cadcells::datastructures::Delineation::add_poly_← nullified ( PolyRef poly ) [inline]`

#### 0.15.153.3.3 add\_root() `void smtrat::cadcells::datastructures::Delineation::add_root ( RAN root, TaggedIndexedRoot tagged_root ) [inline]`

```
0.15.153.3.4 delineate_cell() auto smtrat::cadcells::datastructures::Delineation::delineate_cell()
(
 const RAN & sample) const [inline]
```

Computes the bounds of the interval around the given sample w.r.t.  
this delineation.

```
0.15.153.3.5 empty() bool smtrat::cadcells::datastructures::Delineation::empty () const [inline]
```

```
0.15.153.3.6 nonzero() const auto& smtrat::cadcells::datastructures::Delineation::nonzero () const [inline]
```

The set of polynomials without roots.

```
0.15.153.3.7 nullified() const auto& smtrat::cadcells::datastructures::Delineation::nullified () const [inline]
```

The set of nullified polynomials.

```
0.15.153.3.8 roots() const auto& smtrat::cadcells::datastructures::Delineation::roots () const [inline]
```

Returns the underlying root map, which is a set of real algebraic numbers to indexed root expressions.

## 0.15.153.4 Friends And Related Function Documentation

```
0.15.153.4.1 DelineationInterval friend class DelineationInterval [friend]
```

## 0.15.154 smtrat::cadcells::datastructures::DelineationInterval Class Reference

An interval of a delineation.

```
#include <delineation.h>
```

### Public Member Functions

- bool [is\\_section](#) () const
- bool [is\\_sector](#) () const
- const auto & [lower](#) () const
- bool [lower\\_unbounded](#) () const
- bool [lower\\_strict](#) () const
- const auto & [upper](#) () const
- bool [upper\\_unbounded](#) () const
- bool [upper\\_strict](#) () const
- bool [contains](#) (const RAN &sample) const

### Friends

- class [Delineation](#)

### 0.15.154.1 Detailed Description

An interval of a delineation.

### 0.15.154.2 Member Function Documentation

**0.15.154.2.1 `contains()`** `bool smtrat::cadcells::datastructures::DelineationInterval::contains (`  
`const RAN & sample ) const [inline]`

**0.15.154.2.2 `is_section()`** `bool smtrat::cadcells::datastructures::DelineationInterval::is_section (`  
`) const [inline]`

**0.15.154.2.3 `is_sector()`** `bool smtrat::cadcells::datastructures::DelineationInterval::is_sector (`  
`) const [inline]`

**0.15.154.2.4 `lower()`** `const auto& smtrat::cadcells::datastructures::DelineationInterval::lower (`  
`) const [inline]`

**0.15.154.2.5 `lower_strict()`** `bool smtrat::cadcells::datastructures::DelineationInterval::lower_←`  
`strict ( ) const [inline]`

**0.15.154.2.6 `lower_unbounded()`** `bool smtrat::cadcells::datastructures::DelineationInterval::lower_←`  
`::lower_unbounded ( ) const [inline]`

**0.15.154.2.7 `upper()`** `const auto& smtrat::cadcells::datastructures::DelineationInterval::upper (`  
`) const [inline]`

**0.15.154.2.8 `upper_strict()`** `bool smtrat::cadcells::datastructures::DelineationInterval::upper_←`  
`strict ( ) const [inline]`

**0.15.154.2.9 `upper_unbounded()`** `bool smtrat::cadcells::datastructures::DelineationInterval::upper_←`  
`::upper_unbounded ( ) const [inline]`

### 0.15.154.3 Friends And Related Function Documentation

**0.15.154.3.1 `Delineation`** `friend class Delineation [friend]`

## 0.15.155 `smtrat::mcsat::icp::Dependencies` Class Reference

```
#include <Dependencies.h>
```

### Public Member Functions

- template<typename Contractor >  
  `void add (const Contractor &c)`
- `void add (carl::Variable v, const FormulaT &c, const std::vector< carl::Variable > &used)`
- `std::vector< FormulaT > get (carl::Variable v, bool negate=true) const`

### 0.15.155.1 Member Function Documentation

```
0.15.155.1.1 add() [1/2] void smtrat::mcsat::icp::Dependencies::add (
 carl::Variable v,
 const FormulaT & c,
 const std::vector< carl::Variable > & used) [inline]
```

```
0.15.155.1.2 add() [2/2] template<typename Contractor>
void smtrat::mcsat::icp::Dependencies::add (
 const Contractor & c) [inline]
```

```
0.15.155.1.3 get() std::vector<FormulaT> smtrat::mcsat::icp::Dependencies::get (
 carl::Variable v,
 bool negate = true) const [inline]
```

## 0.15.156 smtrat::cadcells::datastructures::DerivationRef< Properties > Class Template Reference

A reference to a derivation, which is either.

```
#include <derivation.h>
```

### Public Member Functions

- `DerivationRef (const BaseDerivationRef< Properties > &data)`
- `DerivationRef (const DelineatedDerivationRef< Properties > &data)`
- `DerivationRef (const SampledDerivationRef< Properties > &data)`
- `bool is_null () const`
- `bool is_sampled () const`
- `auto & base_ref ()`
- `auto & delineated_ref ()`
- `auto & sampled_ref ()`
- `const auto & base_ref () const`
- `const auto & delineated_ref () const`
- `const auto & sampled_ref () const`
- `auto & base ()`
- `auto & delineated ()`
- `auto & sampled ()`
- `const auto & base () const`
- `const auto & delineated () const`
- `const auto & sampled () const`

### Friends

- `template<typename P >`  
`bool operator==(const DerivationRef< P > &lhs, const DerivationRef< P > &rhs)`
- `template<typename P >`  
`bool operator<(const DerivationRef< P > &lhs, const DerivationRef< P > &rhs)`

### 0.15.156.1 Detailed Description

```
template<typename Properties>
class smtrat::cadcells::datastructures::DerivationRef< Properties >
```

A reference to a derivation, which is either.

- a `BaseDerivation` (a set of properties for each level)

- a **DelineatedDerivation** (a **BaseDerivation** with an underlying **SampledDerivation** on which the properties can be delineated)
- a **SampledDerivation** (a **DelineatedDerivation** with an underlying **SampledDerivation** plus a sample on the current level such that a delineated interval can be computed)

**0.15.156.1.1 Memory management** Memory management is based on std::shared\_ptr. A **SampledDerivation** holds a shared pointer to a **DelineatedDerivation**, which in turn holds a shared pointer to **BaseDerivation**. The **BaseDerivation** holds a **DerivationRef** to the underlying derivation. Note that not all combinations are legal; i.e. the underlying derivation of a delineated derivation must be a sampled derivation.

## 0.15.156.2 Constructor & Destructor Documentation

**0.15.156.2.1 DerivationRef() [1/3]** template<typename Properties >  
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (const BaseDerivationRef< Properties > & data ) [inline]

**0.15.156.2.2 DerivationRef() [2/3]** template<typename Properties >  
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (const DelineatedDerivationRef< Properties > & data ) [inline]

**0.15.156.2.3 DerivationRef() [3/3]** template<typename Properties >  
smtrat::cadcells::datastructures::DerivationRef< Properties >::DerivationRef (const SampledDerivationRef< Properties > & data ) [inline]

## 0.15.156.3 Member Function Documentation

**0.15.156.3.1 base() [1/2]** template<typename Properties >  
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base ( ) [inline]

**0.15.156.3.2 base() [2/2]** template<typename Properties >  
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base ( ) const [inline]

**0.15.156.3.3 base\_ref() [1/2]** template<typename Properties >  
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base\_ref ( ) [inline]

**0.15.156.3.4 base\_ref() [2/2]** template<typename Properties >  
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::base\_ref ( ) const [inline]

**0.15.156.3.5 delineated() [1/2]** template<typename Properties >  
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated ( ) [inline]

**0.15.156.3.6 delineated()** [2/2] template<typename Properties >  
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated ( )  
const [inline]

**0.15.156.3.7 delineated\_ref()** [1/2] template<typename Properties >  
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated\_ref ( ) [inline]

**0.15.156.3.8 delineated\_ref()** [2/2] template<typename Properties >  
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::delineated\_ref ( )  
const [inline]

**0.15.156.3.9 is\_null()** template<typename Properties >  
bool smtrat::cadcells::datastructures::DerivationRef< Properties >::is\_null ( ) const [inline]

**0.15.156.3.10 is\_sampled()** template<typename Properties >  
bool smtrat::cadcells::datastructures::DerivationRef< Properties >::is\_sampled ( ) const  
[inline]

**0.15.156.3.11 sampled()** [1/2] template<typename Properties >  
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled ( ) [inline]

**0.15.156.3.12 sampled()** [2/2] template<typename Properties >  
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled ( ) const  
[inline]

**0.15.156.3.13 sampled\_ref()** [1/2] template<typename Properties >  
auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled\_ref ( ) [inline]

**0.15.156.3.14 sampled\_ref()** [2/2] template<typename Properties >  
const auto& smtrat::cadcells::datastructures::DerivationRef< Properties >::sampled\_ref ( )  
const [inline]

#### 0.15.156.4 Friends And Related Function Documentation

**0.15.156.4.1 operator<** template<typename Properties >  
template<typename P >  
bool operator< (  
 const DerivationRef< P > & lhs,  
 const DerivationRef< P > & rhs ) [friend]

**0.15.156.4.2 operator==** template<typename Properties >  
template<typename P >  
bool operator== (  
 const DerivationRef< P > & lhs,  
 const DerivationRef< P > & rhs ) [friend]

## 0.15.157 smtrat::mcsat::onecellcad::recursive::DontCoverNullification Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr bool `cover_nullification` = false

### 0.15.157.1 Field Documentation

**0.15.157.1.1 cover\_nullification** constexpr bool smtrat::mcsat::onecellcad::recursive::DontCoverNullification::cover\_nullification = false [static], [constexpr]

## 0.15.158 smtrat::cad::debug::DotSubgraph Struct Reference

```
#include <DotHelper.h>
```

### Public Member Functions

- `DotSubgraph` (const std::string &n)
- void `add` (const std::string &n)

### Data Fields

- std::string `name`
- std::vector< std::string > `nodes`

### Friends

- std::ostream & `operator<<` (std::ostream &os, const `DotSubgraph` &dsg)

### 0.15.158.1 Constructor & Destructor Documentation

**0.15.158.1.1 DotSubgraph()** smtrat::cad::debug::DotSubgraph::DotSubgraph (const std::string & n) [inline]

### 0.15.158.2 Member Function Documentation

**0.15.158.2.1 add()** void smtrat::cad::debug::DotSubgraph::add (const std::string & n) [inline]

### 0.15.158.3 Friends And Related Function Documentation

**0.15.158.3.1 operator<<** std::ostream& operator<< (std::ostream & os, const `DotSubgraph` & dsg ) [friend]

### 0.15.158.4 Field Documentation

**0.15.158.4.1 name** std::string smtrat::cad::debug::DotSubgraph::name

**0.15.158.4.2 nodes** std::vector<std::string> smtrat::cad::debug::DotSubgraph::nodes

## 0.15.159 Minisat::DoubleOption Class Reference

```
#include <Options.h>
```

### Public Member Functions

- **DoubleOption** (const char \*c, const char \*n, const char \*d, double def=double(), [DoubleRange r=DoubleRange\(-HUGE\\_VAL, false, HUGE\\_VAL, false\)](#))
- **operator double** (void) const
- **DoubleOption & operator=** (double x)
- virtual void **help** (bool verbose=false)

### Static Protected Member Functions

- static **vec< Option \* > & getOptionList** ()
- static const char \*& **getUsageString** ()
- static const char \*& **getHelpPrefixString** ()

### Protected Attributes

- **DoubleRange range**
- double **value**
- const char \* **name**
- const char \* **description**
- const char \* **category**
- const char \* **type\_name**

## 0.15.159.1 Constructor & Destructor Documentation

**0.15.159.1.1 DoubleOption()** Minisat::DoubleOption::DoubleOption (   
     const char \* c,  
     const char \* n,  
     const char \* d,  
     double def = double(),  
     [DoubleRange r = DoubleRange\( -HUGE\\_VAL, false, HUGE\\_VAL, false \)](#) ) [inline]

## 0.15.159.2 Member Function Documentation

**0.15.159.2.1 getHelpPrefixString()** static const char\*& Minisat::Option::getHelpPrefixString ( )  
[inline], [static], [protected], [inherited]

**0.15.159.2.2 getOptionList()** static **vec<Option\*>&** Minisat::Option::getOptionList ( ) [inline],  
[static], [protected], [inherited]

**0.15.159.2.3 getUsageString()** static const char\*& Minisat::Option::getUsageString ( ) [inline],  
[static], [protected], [inherited]

**0.15.159.2.4 `help()`** `virtual void Minisat::DoubleOption::help (`  
    `bool verbose = false ) [inline], [virtual]`  
Implements [Minisat::Option](#).

**0.15.159.2.5 `operator double()`** `Minisat::DoubleOption::operator double (`  
    `void ) const [inline]`

**0.15.159.2.6 `operator=()`** `DoubleOption& Minisat::DoubleOption::operator= (`  
    `double x ) [inline]`

### 0.15.159.3 Field Documentation

**0.15.159.3.1 `category`** `const char* Minisat::Option::category [protected], [inherited]`

**0.15.159.3.2 `description`** `const char* Minisat::Option::description [protected], [inherited]`

**0.15.159.3.3 `name`** `const char* Minisat::Option::name [protected], [inherited]`

**0.15.159.3.4 `range`** `DoubleRange Minisat::DoubleOption::range [protected]`

**0.15.159.3.5 `type_name`** `const char* Minisat::Option::type_name [protected], [inherited]`

**0.15.159.3.6 `value`** `double Minisat::DoubleOption::value [protected]`

## 0.15.160 Minisat::DoubleRange Struct Reference

#include <Options.h>

### Public Member Functions

- [DoubleRange](#) (double b, bool binc, double e, bool einc)

### Data Fields

- double [begin](#)
- double [end](#)
- bool [begin\\_inclusive](#)
- bool [end\\_inclusive](#)

### 0.15.160.1 Constructor & Destructor Documentation

**0.15.160.1.1 `DoubleRange()`** `Minisat::DoubleRange::DoubleRange (`  
    `double b,`  
    `bool binc,`  
    `double e,`  
    `bool einc ) [inline]`

### 0.15.160.2 Field Documentation

**0.15.160.2.1 begin** double Minisat::DoubleRange::begin

**0.15.160.2.2 begin\_inclusive** bool Minisat::DoubleRange::begin\_inclusive

**0.15.160.2.3 end** double Minisat::DoubleRange::end

**0.15.160.2.4 end\_inclusive** bool Minisat::DoubleRange::end\_inclusive

## 0.15.161 smtrat::expression::simplifier::DuplicateSimplifier Struct Reference

```
#include <DuplicateSimplifier.h>
```

### Public Member Functions

- const ExpressionContent \* simplify (const NaryExpression &expr) const
- const ExpressionContent \* operator() (const carl::Variable &expr) const
- const ExpressionContent \* operator() (const ITEExpression &expr) const
- const ExpressionContent \* operator() (const QuantifierExpression &expr) const
- const ExpressionContent \* operator() (const UnaryExpression &expr) const
- const ExpressionContent \* operator() (const BinaryExpression &expr) const
- const ExpressionContent \* operator() (const NaryExpression &expr) const
- const ExpressionContent \* operator() (const ExpressionContent \*\_ec) const

### Protected Member Functions

- virtual const ExpressionContent \* simplify (const carl::Variable &) const
- virtual const ExpressionContent \* simplify (const ITEExpression &) const
- virtual const ExpressionContent \* simplify (const QuantifierExpression &) const
- virtual const ExpressionContent \* simplify (const UnaryExpression &) const
- virtual const ExpressionContent \* simplify (const BinaryExpression &) const

### 0.15.161.1 Member Function Documentation

**0.15.161.1.1 operator()()** [1/7] const ExpressionContent\* smtrat::expression::simplifier::Base<-Simplifier::operator() (const BinaryExpression & expr ) const [inline], [inherited]

**0.15.161.1.2 operator()()** [2/7] const ExpressionContent\* smtrat::expression::simplifier::Base<-Simplifier::operator() (const carl::Variable & expr ) const [inline], [inherited]

**0.15.161.1.3 operator()()** [3/7] const ExpressionContent\* smtrat::expression::simplifier::Base<-Simplifier::operator() (const ExpressionContent \* \_ec ) const [inline], [inherited]

**0.15.161.1.4 operator() [4/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→ Simplifier::operator() (

```
 const ITEEExpression & expr) const [inline], [inherited]
```

**0.15.161.1.5 operator() [5/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→ Simplifier::operator() (

```
 const NaryExpression & expr) const [inline], [inherited]
```

**0.15.161.1.6 operator() [6/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→ Simplifier::operator() (

```
 const QuantifierExpression & expr) const [inline], [inherited]
```

**0.15.161.1.7 operator() [7/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→ Simplifier::operator() (

```
 const UnaryExpression & expr) const [inline], [inherited]
```

**0.15.161.1.8 simplify() [1/6]** virtual const `ExpressionContent*` smrat::expression::simplifier::← BaseSimplifier::simplify (

```
 const BinaryExpression &) const [inline], [protected], [virtual], [inherited]
```

**0.15.161.1.9 simplify() [2/6]** virtual const `ExpressionContent*` smrat::expression::simplifier::← BaseSimplifier::simplify (

```
 const carl::Variable &) const [inline], [protected], [virtual], [inherited]
```

**0.15.161.1.10 simplify() [3/6]** virtual const `ExpressionContent*` smrat::expression::simplifier::← ::BaseSimplifier::simplify (

```
 const ITEEExpression &) const [inline], [protected], [virtual], [inherited]
```

**0.15.161.1.11 simplify() [4/6]** const `ExpressionContent*` smrat::expression::simplifier::Duplicate<→ Simplifier::simplify (

```
 const NaryExpression & expr) const [inline], [virtual]
```

Reimplemented from `smrat::expression::simplifier::BaseSimplifier`.

**0.15.161.1.12 simplify() [5/6]** virtual const `ExpressionContent*` smrat::expression::simplifier::← ::BaseSimplifier::simplify (

```
 const QuantifierExpression &) const [inline], [protected], [virtual], [inherited]
```

**0.15.161.1.13 simplify() [6/6]** virtual const `ExpressionContent*` smrat::expression::simplifier::← ::BaseSimplifier::simplify (

```
 const UnaryExpression &) const [inline], [protected], [virtual], [inherited]
```

Reimplemented in `smrat::expression::simplifier::NegationSimplifier`.

## 0.15.162 smrat::DynamicPriorityQueue< T, Compare > Class Template Reference

```
#include <DynamicPriorityQueue.h>
```

## Public Types

- using `value_type` = typename Sequence::value\_type

## Public Member Functions

- `DynamicPriorityQueue` (const Compare & \_comp=Compare())
- `DynamicPriorityQueue` (Sequence && \_seq, const Compare & \_comp=Compare())
- void `fix` ()
- bool `empty` () const
- std::size\_t `size` () const
- const T & `top` () const
- void `push` (const T &t)
- void `push` (T &&t)
- template<typename... Args>  
void `emplace` (Args &&... args)
- void `pop` ()
- T `popTop` ()

## Protected Attributes

- Sequence `c`
- Compare `comp`

### 0.15.162.1 Member Typedef Documentation

```
0.15.162.1.1 value_type template<typename T , typename Compare = std::less<T>>
using smtrat::DynamicPriorityQueue< T, Compare >::value_type = typename Sequence::value_type
```

### 0.15.162.2 Constructor & Destructor Documentation

```
0.15.162.2.1 DynamicPriorityQueue() [1/2] template<typename T , typename Compare = std::less<↔
T>>
smtrat::DynamicPriorityQueue< T, Compare >::DynamicPriorityQueue (
 const Compare & _comp = Compare()) [inline], [explicit]
```

```
0.15.162.2.2 DynamicPriorityQueue() [2/2] template<typename T , typename Compare = std::less<↔
T>>
smtrat::DynamicPriorityQueue< T, Compare >::DynamicPriorityQueue (
 Sequence && _seq,
 const Compare & _comp = Compare()) [inline], [explicit]
```

### 0.15.162.3 Member Function Documentation

```
0.15.162.3.1 emplace() template<typename T , typename Compare = std::less<T>>
template<typename... Args>
void smtrat::DynamicPriorityQueue< T, Compare >::emplace (
 Args &&... args) [inline]
```

**0.15.162.3.2 `empty()`** template<typename T , typename Compare = std::less<T>>  
bool smtrat::DynamicPriorityQueue< T, Compare >::empty ( ) const [inline]

**0.15.162.3.3 `fix()`** template<typename T , typename Compare = std::less<T>>  
void smtrat::DynamicPriorityQueue< T, Compare >::fix ( ) [inline]

**0.15.162.3.4 `pop()`** template<typename T , typename Compare = std::less<T>>  
void smtrat::DynamicPriorityQueue< T, Compare >::pop ( ) [inline]

**0.15.162.3.5 `popTop()`** template<typename T , typename Compare = std::less<T>>  
T smtrat::DynamicPriorityQueue< T, Compare >::popTop ( ) [inline]

**0.15.162.3.6 `push() [1/2]`** template<typename T , typename Compare = std::less<T>>  
void smtrat::DynamicPriorityQueue< T, Compare >::push (  
 const T & t ) [inline]

**0.15.162.3.7 `push() [2/2]`** template<typename T , typename Compare = std::less<T>>  
void smtrat::DynamicPriorityQueue< T, Compare >::push (  
 T && t ) [inline]

**0.15.162.3.8 `size()`** template<typename T , typename Compare = std::less<T>>  
std::size\_t smtrat::DynamicPriorityQueue< T, Compare >::size ( ) const [inline]

**0.15.162.3.9 `top()`** template<typename T , typename Compare = std::less<T>>  
const T& smtrat::DynamicPriorityQueue< T, Compare >::top ( ) const [inline]

#### 0.15.162.4 Field Documentation

**0.15.162.4.1 `c`** template<typename T , typename Compare = std::less<T>>  
Sequence smtrat::DynamicPriorityQueue< T, Compare >::c [protected]

**0.15.162.4.2 `comp`** template<typename T , typename Compare = std::less<T>>  
Compare smtrat::DynamicPriorityQueue< T, Compare >::comp [protected]

### 0.15.163 smtrat::dynarray< T > Class Template Reference

dynarray is similar to std::vector, but has only a reduced interface.

```
#include <alloc.h>
```

#### Public Types

- `typedef T value_type`
- `typedef T * iterator`
- `typedef const T * const_iterator`
- `typedef T & reference`
- `typedef const T & const_reference`

- `typedef T * pointer`
- `typedef const T * const_pointer`

### Public Member Functions

- `dynarray ()`
- `dynarray (std::size_t initialCap)`
- `~dynarray ()`
- `dynarray (const dynarray &)=delete`
- `dynarray & operator= (const dynarray &)=delete`
- `dynarray (dynarray &&other) NOEXCEPT`
- `dynarray & operator= (dynarray &&other) NOEXCEPT`
- `iterator begin () NOEXCEPT`
- `iterator end () NOEXCEPT`
- `const_iterator begin () const NOEXCEPT`
- `const_iterator end () const NOEXCEPT`
- `bool empty () const NOEXCEPT`
- `std::size_t size () const NOEXCEPT`
- `std::size_t capacity () const NOEXCEPT`
- `void remove (const T &value)`
- template<typename... Args>  
  `T & emplace_back (Args &&... args)`
- `void swap (dynarray< T > &other) NOEXCEPT`
- `void swap (dynarray< T > &&other) NOEXCEPT`
- `void consume (dynarray< T > &&other)`
- `T & front () NOEXCEPT`
- `T & back () NOEXCEPT`
- `void pop_back () NOEXCEPT`
- `reference operator[] (std::size_t index) NOEXCEPT`
- `const_reference operator[] (std::size_t index) const NOEXCEPT`
- `void clear () NOEXCEPT`
- `void reserve (std::size_t minCap)`

#### 0.15.163.1 Detailed Description

```
template<typename T>
class smtrat::dynarray< T >
```

`dynarray` is similar to `std::vector`, but has only a reduced interface.

Moreover, it releases its memory to a `dynarray_allocator` in its destructor and supports an optimized `consume` method to move-add all elements from some `dynarray` to another, and also a `swap-to-end` remove method. It is to be used in contexts where dynamic arrays have to be allocated and freed relatively frequently.

#### 0.15.163.2 Member Typedef Documentation

```
0.15.163.2.1 const_iterator template<typename T >
typedef const T* smtrat::dynarray< T >::const_iterator
```

```
0.15.163.2.2 const_pointer template<typename T >
typedef const T* smtrat::dynarray< T >::const_pointer
```

**0.15.163.2.3 const\_reference** template<typename T >  
typedef const T& smtrat::dynarray< T >::const\_reference

**0.15.163.2.4 iterator** template<typename T >  
typedef T\* smtrat::dynarray< T >::iterator

**0.15.163.2.5 pointer** template<typename T >  
typedef T\* smtrat::dynarray< T >::pointer

**0.15.163.2.6 reference** template<typename T >  
typedef T& smtrat::dynarray< T >::reference

**0.15.163.2.7 value\_type** template<typename T >  
typedef T smtrat::dynarray< T >::value\_type

### 0.15.163.3 Constructor & Destructor Documentation

**0.15.163.3.1 dynarray() [1/4]** template<typename T >  
smtrat::dynarray< T >::dynarray ( ) [inline]

**0.15.163.3.2 dynarray() [2/4]** template<typename T >  
smtrat::dynarray< T >::dynarray ( std::size\_t initialCap ) [inline]

**0.15.163.3.3 ~dynarray()** template<typename T >  
smtrat::dynarray< T >::~dynarray ( ) [inline]

**0.15.163.3.4 dynarray() [3/4]** template<typename T >  
smtrat::dynarray< T >::dynarray ( const dynarray< T > & ) [delete]

**0.15.163.3.5 dynarray() [4/4]** template<typename T >  
smtrat::dynarray< T >::dynarray ( dynarray< T > && other ) [inline]

### 0.15.163.4 Member Function Documentation

**0.15.163.4.1 back()** template<typename T >  
T& smtrat::dynarray< T >::back ( ) [inline]

**0.15.163.4.2 begin() [1/2]** template<typename T >  
const\_iterator smtrat::dynarray< T >::begin ( ) const [inline]

**0.15.163.4.3 `begin()` [2/2]** template<typename T >  
iterator smtrat::dynarray< T >::begin ( ) const [inline]

**0.15.163.4.4 `capacity()`** template<typename T >  
std::size\_t smtrat::dynarray< T >::capacity ( ) const [inline]

**0.15.163.4.5 `clear()`** template<typename T >  
void smtrat::dynarray< T >::clear ( ) const [inline]

**0.15.163.4.6 `consume()`** template<typename T >  
void smtrat::dynarray< T >::consume (   
 dynarray< T > && other ) const [inline]

**0.15.163.4.7 `emplace_back()`** template<typename T >  
template<typename... Args>  
T& smtrat::dynarray< T >::emplace\_back (   
 Args &&... args ) const [inline]

**0.15.163.4.8 `empty()`** template<typename T >  
bool smtrat::dynarray< T >::empty ( ) const [inline]

**0.15.163.4.9 `end() [1/2]`** template<typename T >  
const\_iterator smtrat::dynarray< T >::end ( ) const [inline]

**0.15.163.4.10 `end() [2/2]`** template<typename T >  
iterator smtrat::dynarray< T >::end ( ) const [inline]

**0.15.163.4.11 `front()`** template<typename T >  
T& smtrat::dynarray< T >::front ( ) const [inline]

**0.15.163.4.12 `operator=() [1/2]`** template<typename T >  
dynarray& smtrat::dynarray< T >::operator= (   
 const dynarray< T > & ) const [delete]

**0.15.163.4.13 `operator=() [2/2]`** template<typename T >  
dynarray& smtrat::dynarray< T >::operator= (   
 dynarray< T > && other ) const [inline]

**0.15.163.4.14 `operator[]()` [1/2]** template<typename T >  
const\_reference smtrat::dynarray< T >::operator[] (   
 std::size\_t index ) const [inline]

**0.15.163.4.15 operator[]( ) [2/2]** template<typename T >  
reference smtrat::dynarray< T >::operator[] ( std::size\_t index ) [inline]

**0.15.163.4.16 pop\_back()** template<typename T >  
void smtrat::dynarray< T >::pop\_back ( ) [inline]

**0.15.163.4.17 remove()** template<typename T >  
void smtrat::dynarray< T >::remove ( const T & value ) [inline]

**0.15.163.4.18 reserve()** template<typename T >  
void smtrat::dynarray< T >::reserve ( std::size\_t minCap ) [inline]

**0.15.163.4.19 size()** template<typename T >  
std::size\_t smtrat::dynarray< T >::size ( ) const [inline]

**0.15.163.4.20 swap() [1/2]** template<typename T >  
void smtrat::dynarray< T >::swap ( dynarray< T > && other ) [inline]

**0.15.163.4.21 swap() [2/2]** template<typename T >  
void smtrat::dynarray< T >::swap ( dynarray< T > & other ) [inline]

## 0.15.164 smtrat::dynarray\_allocator< T > Class Template Reference

#include <alloc.h>

### Friends

- class dynarray< T >

### 0.15.164.1 Friends And Related Function Documentation

**0.15.164.1.1 dynarray< T >** template<typename T >  
friend class dynarray< T > [friend]

## 0.15.165 smtrat::datastructures::DynHeap< KeyType, Compare, WeightType > Class Template Reference

#include <DynHeap.h>

## Public Member Functions

- `DynHeap ()`
- `DynHeap (const std::vector< std::pair< KeyType, WeightType >> &elements)`
- `bool is_empty () const`
- `std::size_t size () const`
- `void insert (const KeyType &pKey, const WeightType pWeight)`
- `KeyType extractMin ()`
- `void update (const KeyType &pKey, const WeightType pWeight)`
- `void clear ()`

### 0.15.165.1 Constructor & Destructor Documentation

**0.15.165.1.1 DynHeap() [1/2]** `template<typename KeyType , class Compare = less<KeyType>, typename WeightType = std::size_t>`  
`smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::DynHeap ( ) [inline], [explicit]`

**0.15.165.1.2 DynHeap() [2/2]** `template<typename KeyType , class Compare = less<KeyType>, typename WeightType = std::size_t>`  
`smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::DynHeap (`  
`const std::vector< std::pair< KeyType, WeightType >> & elements ) [inline],`  
`[explicit]`

### 0.15.165.2 Member Function Documentation

**0.15.165.2.1 clear()** `template<typename KeyType , class Compare = less<KeyType>, typename WeightType = std::size_t>`  
`void smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::clear ( ) [inline]`

**0.15.165.2.2 extractMin()** `template<typename KeyType , class Compare = less<KeyType>, typename WeightType = std::size_t>`  
`KeyType smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::extractMin ( ) [inline]`

**0.15.165.2.3 insert()** `template<typename KeyType , class Compare = less<KeyType>, typename WeightType = std::size_t>`  
`void smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::insert (`  
`const KeyType & pKey,`  
`const WeightType pWeight ) [inline]`

**0.15.165.2.4 is\_empty()** `template<typename KeyType , class Compare = less<KeyType>, typename WeightType = std::size_t>`  
`bool smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::is_empty ( ) const`  
`[inline]`

**0.15.165.2.5 size()** `template<typename KeyType , class Compare = less<KeyType>, typename WeightType = std::size_t>`  
`std::size_t smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::size ( ) const`  
`[inline]`

```
0.15.165.2.6 update() template<typename KeyType , class Compare = less<KeyType>, typename
WeightType = std::size_t>
void smtrat::datastructures::DynHeap< KeyType, Compare, WeightType >::update (
 const KeyType & pKey,
 const WeightType pWeight) [inline]
```

## 0.15.166 smtrat::cad::ProjectionGlobalInformation::ECData Struct Reference

```
#include <ProjectionInformation.h>
```

### Data Fields

- std::size\_t **level** = 0
- carl::Bitset **polynomials**

#### 0.15.166.1 Field Documentation

**0.15.166.1.1 level** std::size\_t smtrat::cad::ProjectionGlobalInformation::ECData::level = 0

**0.15.166.1.2 polynomials** carl::Bitset smtrat::cad::ProjectionGlobalInformation::ECData::polynomials

## 0.15.167 smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge Struct Reference

Internal type of an edge.

```
#include <ForwardHyperGraph.h>
```

### Public Member Functions

- **Edge** (const EdgeProperty &ep)  
*Constructor from a property.*
- void **addOut** (Vertex v)  
*Add v to the outgoing vertices of this edge.*
- const EdgeProperty & **operator\*** () const  
*Access the property.*
- EdgeProperty & **operator\*** ()  
*Access the property.*
- const EdgeProperty \* **operator->** () const  
*Access the property.*
- EdgeProperty \* **operator->** ()  
*Access the property.*
- const std::vector< Vertex > & **out** () const  
*Retrieve the list of outgoing vertices.*

### Data Fields

- friend **ForwardHyperGraph**

#### 0.15.167.1 Detailed Description

```
template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices = true>
struct smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge
```

Internal type of an edge.

### 0.15.167.2 Constructor & Destructor Documentation

**0.15.167.2.1 Edge()** template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>  
smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::Edge ( const EdgeProperty & ep ) [inline]  
Constructor from a property.

### 0.15.167.3 Member Function Documentation

**0.15.167.3.1 addOut()** template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>  
void smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::addOut ( Vertex v ) [inline]  
Add v to the outgoing vertices of this edge.

**0.15.167.3.2 operator\*() [1/2]** template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>  
EdgeProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::operator\* () [inline]  
Access the property.

**0.15.167.3.3 operator\*() [2/2]** template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>  
const EdgeProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::operator\* () const [inline]  
Access the property.

**0.15.167.3.4 operator->() [1/2]** template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>  
EdgeProperty\* smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::operator-> () [inline]  
Access the property.

**0.15.167.3.5 operator->() [2/2]** template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>  
const EdgeProperty\* smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::operator-> () const [inline]  
Access the property.

**0.15.167.3.6 out()** template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>  
const std::vector<Vertex>& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::out () const [inline]  
Retrieve the list of outgoing vertices.

### 0.15.167.4 Field Documentation

```
0.15.167.4.1 ForwardHyperGraph template<typename VertexProperty , typename EdgeProperty ,
bool UniqueVertices = true>
friend smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Edge::←
ForwardHyperGraph
```

## 0.15.168 **smtrat::EQModule< Settings >::graph\_info::edge\_list\_type< EdgeType >** Struct Template Reference

Implements a list of edges.

```
#include <EQModule.h>
```

### Public Types

- `typedef EdgeType ** iterator`
- `typedef const EdgeType *const * const_iterator`

### Public Member Functions

- `edge_list_type (freelist< EdgeType > &alloc) noexcept`
- `~edge_list_type ()`
- `iterator begin () noexcept`
- `iterator end () noexcept`
- `const_iterator begin () const noexcept`
- `const_iterator end () const noexcept`
- `EdgeType *& operator[] (std::size_t i) noexcept`
- `const EdgeType * operator[] (std::size_t i) const noexcept`
- `std::size_t size () const noexcept`
- `void add (EdgeType *edge)`
- `void remove (EdgeType *edge)`
- `void remove_back ()`
- `void remove_destroy (EdgeType *edge)`

### Friends

- class `graph_info`

### 0.15.168.1 Detailed Description

```
template<typename Settings>
template<typename EdgeType>
struct smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >
```

Implements a list of edges.

This is similar to a vector of edge-pointers, but allocates pointers from a freelist and supports O(1) removal by storing the index of an edge in this list inside the edge.

### 0.15.168.2 Member Typedef Documentation

```
0.15.168.2.1 const_iterator template<typename Settings >
template<typename EdgeType >
typedef const EdgeType* const* smtrat::EQModule< Settings >::graph_info::edge_list_type<
EdgeType >::const_iterator
```

```
0.15.168.2.2 iterator template<typename Settings >
template<typename EdgeType >
typedef EdgeType** smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::iterator
```

### 0.15.168.3 Constructor & Destructor Documentation

```
0.15.168.3.1 edge_list_type() template<typename Settings >
template<typename EdgeType >
smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::edge_list_type (
 freelist< EdgeType > & alloc) [inline], [noexcept]
```

```
0.15.168.3.2 ~edge_list_type() template<typename Settings >
template<typename EdgeType >
smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::~edge_list_type ()
```

### 0.15.168.4 Member Function Documentation

```
0.15.168.4.1 add() template<typename Settings >
template<typename EdgeType >
void smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::add (
 EdgeType * edge) [inline]
```

```
0.15.168.4.2 begin() [1/2] template<typename Settings >
template<typename EdgeType >
const_iterator smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::begin (
) const [inline], [noexcept]
```

```
0.15.168.4.3 begin() [2/2] template<typename Settings >
template<typename EdgeType >
iterator smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::begin ()
[inline], [noexcept]
```

```
0.15.168.4.4 end() [1/2] template<typename Settings >
template<typename EdgeType >
const_iterator smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::end ()
const [inline], [noexcept]
```

```
0.15.168.4.5 end() [2/2] template<typename Settings >
template<typename EdgeType >
iterator smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::end () [inline],
[noexcept]
```

```
0.15.168.4.6 operator[]() [1/2] template<typename Settings >
template<typename EdgeType >
const EdgeType* smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::operator[]
(
 std::size_t i) const [inline], [noexcept]
```

```
0.15.168.4.7 operator[]() [2/2] template<typename Settings >
template<typename EdgeType >
EdgeType* & smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::operator[]
(
 std::size_t i) [inline], [noexcept]
```

```
0.15.168.4.8 remove() template<typename Settings >
template<typename EdgeType >
void smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::remove (
 EdgeType * edge) [inline]
```

```
0.15.168.4.9 remove_back() template<typename Settings >
template<typename EdgeType >
void smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::remove_back ()
[inline]
```

```
0.15.168.4.10 remove_destroy() template<typename Settings >
template<typename EdgeType >
void smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::remove_destroy (
 EdgeType * edge) [inline]
```

```
0.15.168.4.11 size() template<typename Settings >
template<typename EdgeType >
std::size_t smtrat::EQModule< Settings >::graph_info::edge_list_type< EdgeType >::size ()
const [inline], [noexcept]
```

## 0.15.168.5 Friends And Related Function Documentation

```
0.15.168.5.1 graph_info template<typename Settings >
template<typename EdgeType >
friend class graph_info [friend]
```

## 0.15.169 smtrat::ElementWithOrigins< Element, Origin > Class Template Reference

```
#include <IntBlastModule.h>
```

### Public Member Functions

- `ElementWithOrigins` (const `Element` & `_element`)
- `ElementWithOrigins` (const `Element` & `_element`, const `Origin` & `_origin`)
- const `Element` & `element` () const
- const `std::set< Origin >` & `origins` () const
- void `addOrigin` (const `Origin` & `_origin`)
- bool `removeOrigin` (const `Origin` & `_origin`)
- bool `hasOrigins` ()

### 0.15.169.1 Constructor & Destructor Documentation

**0.15.169.1.1 ElementWithOrigins() [1/2]** template<typename Element , typename Origin >  
`smtrat::ElementWithOrigins< Element, Origin >::ElementWithOrigins (`  
 `const Element & _element ) [inline]`

**0.15.169.1.2 ElementWithOrigins() [2/2]** template<typename Element , typename Origin >  
`smtrat::ElementWithOrigins< Element, Origin >::ElementWithOrigins (`  
 `const Element & _element,`  
 `const Origin & _origin ) [inline]`

### 0.15.169.2 Member Function Documentation

**0.15.169.2.1 addOrigin()** template<typename Element , typename Origin >  
`void smtrat::ElementWithOrigins< Element, Origin >::addOrigin (`  
 `const Origin & _origin ) [inline]`

**0.15.169.2.2 element()** template<typename Element , typename Origin >  
`const Element& smtrat::ElementWithOrigins< Element, Origin >::element ( ) const [inline]`

**0.15.169.2.3 hasOrigins()** template<typename Element , typename Origin >  
`bool smtrat::ElementWithOrigins< Element, Origin >::hasOrigins ( ) [inline]`

**0.15.169.2.4 origins()** template<typename Element , typename Origin >  
`const std::set<Origin>& smtrat::ElementWithOrigins< Element, Origin >::origins ( ) const [inline]`

**0.15.169.2.5 removeOrigin()** template<typename Element , typename Origin >  
`bool smtrat::ElementWithOrigins< Element, Origin >::removeOrigin (`  
 `const Origin & _origin ) [inline]`

## 0.15.170 smtrat::EMModule< Settings > Class Template Reference

```
#include <EMModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `EMModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~EMModule ()`
- `Answer checkCore ()`  
`Checks the received formula for consistency.`
- `bool isPreprocessor () const`
- `bool appliedPreprocessing () const`

- `bool add (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `void remove (ModuleInput::const_iterator _subformula)`  
*Removes everything related to the given sub-formula of the received formula.*
- `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
*Checks the received formula for consistency.*
- `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- `virtual Answer runBackends ()`
- `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void updateModel () const`  
*Updates the current assignment into the model.*
- `bool inform (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void deinform (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- `virtual void init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `virtual void updateAllModels ()`  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`  
*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`  
*Sets the priority of this module to get a thread for running its check procedure.*
- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`
- `const std::vector< Model > & allModels () const`
- `const std::vector< FormulaSetT > & infeasibleSubsets () const`
- `const std::vector< Module * > & usedBackends () const`
- `const carl::FastSet< FormulaT > & constraintsToInform () const`
- `const carl::FastSet< FormulaT > & informedConstraints () const`
- `void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`  
*Stores a lemma being a valid formula.*
- `bool hasLemmas ()`  
*Checks whether this module or any of its backends provides any lemmas.*
- `void clearLemmas ()`  
*Deletes all yet found lemmas.*
- `const std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`

- void `receivedFormulaChecked ()`  
*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- virtual std::string `moduleName () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`  
*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `hasValidInfeasibleSubset () const`
- void `checkInfeasibleSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _insubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore = 5`  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches = 0`  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)
 

*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)
 

*The inverse of informing about a constraint.*
- virtual bool `addCore` (`ModuleInput::const_iterator` formula)
 

*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (`ModuleInput::const_iterator` formula)
 

*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const
 

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const
 

*Clears the assignment, if any was found.*
- void `clearModels` () const
 

*Clears all assignments, if any was found.*
- void `cleanModel` () const
 

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
 

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const
 

*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()

- Copies the infeasible subsets of the passed formula.
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
  - Get the infeasible subsets the given backend provides.
- const `Model` & `backendsModel` () const
  - Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void `getBackendsModel` () const
- void `getBackendsAllModels` () const
  - Stores all models of a backend in the list of all models of this module.
- virtual `ModuleInput`::iterator `eraseSubformulaFromPassedFormula` (`ModuleInput`::iterator \_subformula, bool \_ignoreOrigins=false)
  - Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void `clearPassedFormula` ()
- std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const
  - Merges the two vectors of sets into the first one.
- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename `Poly`::`PolyType` &\_branchingPolynomial, const `Rational` &\_branchingValue) const
  - Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &&\_premise=std::vector< `FormulaT` >())
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- void `splitUnequalConstraint` (const `FormulaT` &)
  - Adds the following lemmas for the given constraint  $p \neq 0$ .
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
  - Adds a formula to the InformationRelevantFormula.
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
  - Gets all InformationRelevantFormulas.
- bool `isLemmaLevel` (`LemmaLevel` level)
  - Checks if current lemma level is greater or equal to given level.

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
  - Checks whether there is no variable assigned by both models.

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`
  - Stores the infeasible subsets.
- `Manager` \*const `mpManager`
  - A reference to the manager.
- `Model` `mModel`

- Stores the assignment of the current satisfiable result, if existent.*
- std::vector< Model > mAllModels  
*Stores all satisfying assignments.*
  - bool mModelComputed  
*True, if the model has already been computed.*
  - bool mFinalCheck  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
  - bool mFullCheck  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
  - carl::Variable mObjectiveVariable  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
  - std::vector< TheoryPropagation > mTheoryPropagations
  - std::atomic< Answer > mSolverState  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
  - std::atomic\_bool \* mBackendsFoundAnswer  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
  - Conditionals mFoundAnswer  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
  - std::vector< Module \* > mUsedBackends  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
  - std::vector< Module \* > mAllBackends  
*The backends of this module which have been used.*
  - std::vector< Lemma > mLemmas  
*Stores the lemmas being valid formulas this module or its backends made.*
  - ModuleInput::iterator mFirstSubformulaToPass  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
  - carl::FastSet< FormulaT > mConstraintsToInform  
*Stores the constraints which the backends must be informed about.*
  - carl::FastSet< FormulaT > mInformedConstraints  
*Stores the position of the first constraint of which no backend has been informed about.*
  - ModuleInput::const\_iterator mFirstUncheckedReceivedSubformula  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
  - unsigned mSmallerMusesCheckCounter  
*Counter used for the generation of the smt2 files to check for smaller muses.*
  - std::vector< std::size\_t > mVariableCounters  
*Maps variables to the number of their occurrences.*

### 0.15.170.1 Member Typedef Documentation

**0.15.170.1.1 SettingsType** template<typename Settings >  
typedef Settings smtrat::EMModule< Settings >::SettingsType

### 0.15.170.2 Member Enumeration Documentation

**0.15.170.2.1 LemmaType** enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.170.3 Constructor & Destructor Documentation

**0.15.170.3.1 `EMModule()`** template<typename Settings>  
`smtrat::EMModule< Settings >::EMModule(`  
`const ModuleInput * _formula,`  
`Conditionals & _conditionals,`  
`Manager * _manager = NULL )`

**0.15.170.3.2 `~EMModule()`** template<typename Settings>  
`smtrat::EMModule< Settings >::~EMModule( )`

### 0.15.170.4 Member Function Documentation

**0.15.170.4.1 `add()`** bool smtrat::PModule::add(

`ModuleInput::const_iterator _subformula`) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.170.4.2 `addConstraintToInform()`** void smtrat::Module::addConstraintToInform(

`const FormulaT & _constraint`) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in `smtrat::SATModule< Settings >`.

**0.15.170.4.3 `addCore()`** virtual bool smtrat::Module::addCore(

`ModuleInput::const_iterator formula`) [inline], [protected], [virtual], [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.170.4.4 addInformationRelevantFormula()** void smrat::Module::addInformationRelevantFormula (

const [FormulaT](#) & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                         |                |
|-------------------------|----------------|
| <a href="#">formula</a> | Formula to add |
|-------------------------|----------------|

**0.15.170.4.5 addLemma()** void smrat::Module::addLemma (

const [FormulaT](#) & \_lemma,

const [LemmaType](#) & \_lt = [LemmaType::NORMAL](#),

const [FormulaT](#) & \_preferredFormula = [FormulaT\( carl::FormulaType::TRUE \)](#) ) [inline],

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                                   |                                                                       |
|-----------------------------------|-----------------------------------------------------------------------|
| <a href="#">_lemma</a>            | The eduction/lemma to store.                                          |
| <a href="#">_lt</a>               | The type of the lemma.                                                |
| <a href="#">_preferredFormula</a> | Hint for the next decision, which formula should be assigned to true. |

**0.15.170.4.6 addOrigin()** void smrat::Module::addOrigin (

[ModuleInput::iterator](#) \_formula,

const [FormulaT](#) & \_origin ) [inline], [protected], [inherited]

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                          |                                                           |
|--------------------------|-----------------------------------------------------------|
| <a href="#">_formula</a> | The passed formula to set the origins for.                |
| <a href="#">_origin</a>  | A set of formulas in the received formula of this module. |

**0.15.170.4.7 addReceivedSubformulaToPassedFormula()** std::pair<[ModuleInput::iterator](#), bool> smrat::Module::addReceivedSubformulaToPassedFormula (

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

```
0.15.170.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat<-
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.170.4.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat<-
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.170.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.170.4.11 `allModels()`** `const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]`

**Returns**

All satisfying assignments, if existent.

**0.15.170.4.12 `anAnswerFound()`** `bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.170.4.13 `answerFound()`** `const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]`

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.170.4.14 `appliedPreprocessing()`** `bool smtrat::PModule::appliedPreprocessing () const [inline], [inherited]`

**Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.170.4.15 `backendsModel()`** `const Model & smtrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.170.4.16 branchAt() [1/4]** bool smtrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.170.4.17 branchAt() [2/4]** bool smtrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.170.4.18 branchAt() [3/4]** bool smtrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.170.4.19 branchAt() [4/4]** bool smtrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

**0.15.170.4.20 check()** `Answer smtrat::PModule::check (`  
    `bool _final = false,`  
    `bool _full = true,`  
    `carl::Variable _objective = carl::Variable::NO_VARIABLE ) [virtual], [inherited]`

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.170.4.21 checkCore()** `template<typename Settings >`  
`Answer smtrat::EMModule< Settings >::checkCore ( ) [virtual]`

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.170.4.22 checkInfeasibleSubsetForMinimality()** `void smtrat::Module::checkInfeasibleSubsetForMinimality (`  
    `std::vector< FormulaSetT >::const_iterator _infssubset,`  
    `const std::string & _filename = "smaller_muses",`  
    `unsigned _maxSizeDifference = 1 ) const [inherited]`

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.170.4.23 checkModel()** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.170.4.24 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.170.4.25 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.170.4.26 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.170.4.27 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.170.4.28 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.170.4.29 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.170.4.30 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.170.4.31 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.170.4.32 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.170.4.33 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.170.4.34 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.170.4.35 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.170.4.36 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.170.4.37 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.170.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.170.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.170.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.170.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const` [inline], [inherited]

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.170.4.42 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const` [inline], [inherited]

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.170.4.43 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.170.4.44 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.170.4.45 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.170.4.46 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.170.4.47 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.170.4.48 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.170.4.49 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.170.4.50 getModelEqualities()** std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.170.4.51 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.170.4.52 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.170.4.53 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

#### Parameters

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.170.4.54 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.170.4.55 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.170.4.56 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

#### Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.170.4.57 `id()`** `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.170.4.58 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.170.4.59 `inform()`** `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.170.4.60 `informBackends()`** `void smtrat::Module::informBackends ( const FormulaT & _constraint ) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.170.4.61 `informCore()`** `virtual bool smtrat::Module::informCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.170.4.62 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

#### Returns

The position of the first constraint of which no backend has been informed about.

**0.15.170.4.63 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.170.4.64 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.170.4.65 isLemmaLevel()** `bool smrat::Module::isLemmaLevel( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.170.4.66 isPreprocessor()** `bool smrat::PModule::isPreprocessor( ) const [inline], [inherited]`

#### Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.170.4.67 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas( ) const [inline], [inherited]`

#### Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.170.4.68 merge()** `std::vector< FormulaT > smrat::Module::merge( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.170.4.69 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.170.4.70 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (`

```
const Model & _modelA,
const Model & _modelB) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.170.4.71 `moduleName()`** `virtual std::string smtrat::Module::moduleName () const [inline], [virtual], [inherited]`**Returns**

The name of the given module type as name.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SplitSOSModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LVEModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::GBModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQPreprocessingModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::BEModule< Settings >](#).

**0.15.170.4.72 `objective()`** `carl::Variable smtrat::Module::objective () const [inline], [inherited]`**0.15.170.4.73 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`

```
const FormulaT & _origin) const [protected], [inherited]
```

**0.15.170.4.74 passedFormulaBegin()** `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )` [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.170.4.75 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )` [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.170.4.76 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const` [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.170.4.77 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const` [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.170.4.78 print()** `void smtrat::Module::print ( const std::string & _initiation = "***" ) const` [inherited]

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.170.4.79 printAllModels()** `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.170.4.80 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

## Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.170.4.81 `printModel()`** `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

## Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.170.4.82 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

## Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.170.4.83 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

## Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.170.4.84 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

## Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

## Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.170.4.85 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.170.4.86 `receivedFormulasAsInfeasibleSubset()`** void smtrat::Module::receivedFormulasAsInfeasibleSubset (

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

**0.15.170.4.87 `receivedVariable()`** bool smtrat::Module::receivedVariable (

```
 carl::Variable::Arg _var) const [inline], [inherited]
```

**0.15.170.4.88 `remove()`** void smtrat::PModule::remove (

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.170.4.89 `removeCore()`** virtual void smtrat::Module::removeCore (

```
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICEModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::GBModule< Settings >](#).

**0.15.170.4.90 `removeOrigin()`** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.170.4.91 `removeOrigins()`** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (

```
ModuleInput::iterator _formula,
const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

#### 0.15.170.4.92 rPassedFormula() const ModuleInput& smtrat::Module::rPassedFormula () const [inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

#### 0.15.170.4.93 rReceivedFormula() const ModuleInput& smtrat::Module::rReceivedFormula () const [inline], [inherited]

**Returns**

A reference to the formula passed to this module.

#### 0.15.170.4.94 runBackends() [1/2] virtual Answer smtrat::PModule::runBackends () [inline], [virtual], [inherited]

Reimplemented from [smtrat::Module](#).

#### 0.15.170.4.95 runBackends() [2/2] Answer smtrat::PModule::runBackends (

```
bool _final,
bool _full,
carl::Variable _objective) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

#### 0.15.170.4.96 setId() void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]

Sets this modules unique ID to identify itself.

**Parameters**

|                                                                                                     |                                    |
|-----------------------------------------------------------------------------------------------------|------------------------------------|
|  <code>id</code> | The id to set this module's id to. |
|-----------------------------------------------------------------------------------------------------|------------------------------------|

**0.15.170.4.97 `setThreadPriority()`** `void smtrat::Module::setThreadPriority (`  
    `thread_priority _threadPriority )` [inline], [inherited]  
Sets the priority of this module to get a thread for running its check procedure.

Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.170.4.98 `solverState()`** `Answer smtrat::Module::solverState ( ) const` [inline], [inherited]

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.170.4.99 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint (`  
    `const FormulaT & _unequalConstraint )` [protected], [inherited]

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.170.4.100 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const` [inline], [inherited]

Returns

The priority of this module to get a thread for running its check procedure.

**0.15.170.4.101 `updateAllModels()`** `void smtrat::Module::updateAllModels ( )` [virtual], [inherited]

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.170.4.102 `updateLemmas()`** `void smtrat::Module::updateLemmas ( )` [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.170.4.103 `updateModel()`** `void smtrat::PModule::updateModel ( ) const` [virtual], [inherited]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.170.4.104 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`  
const [inline], [inherited]

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.170.5 Field Documentation

**0.15.170.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.170.5.2 mAllModels** `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

**0.15.170.5.3 mBackendsFoundAnswer** `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.170.5.4 mConstraintsToInform** `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

**0.15.170.5.5 mFinalCheck** `bool smtrat::Module::mFinalCheck [protected], [inherited]`  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.170.5.6 mFirstPosInLastBranches** `std::size_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

**0.15.170.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.170.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.170.5.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.170.5.10 mFullCheck** `bool smtrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.170.5.11 mInfeasibleSubsets** std::vector<[FormulaSetT](#)> smtrat::Module::mInfeasibleSubsets  
[protected], [inherited]  
Stores the infeasible subsets.

**0.15.170.5.12 mInformedConstraints** carl::FastSet<[FormulaT](#)> smtrat::Module::mInformedConstraints  
[protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.170.5.13 mLastBranches** std::vector< [Branching](#) > smtrat::Module::mLastBranches = std::vector<[Branching](#)>([mNumOfBranchVarsToStore](#), [Branching](#)(typename Poly::PolyType(), 0)) [static],  
[inherited]  
Stores the last branches in a cycle buffer.

**0.15.170.5.14 mLemmas** std::vector<[Lemma](#)> smtrat::Module::mLemmas [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.170.5.15 mModel** [Model](#) smtrat::Module::mModel [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.170.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected],  
[inherited]  
True, if the model has already been computed.

**0.15.170.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.170.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected],  
[inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.170.5.19 mOldSplittingVariables** std::vector< [FormulaT](#) > smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.170.5.20 mpManager** [Manager](#)\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.170.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.170.5.22 mSolverState** std::atomic<[Answer](#)> smrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.170.5.23 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smrat::Module::mTheory← Propagations [protected], [inherited]

**0.15.170.5.24 mUsedBackends** std::vector<[Module\\*](#)> smrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.170.5.25 mVariableCounters** std::vector<std::size\_t> smrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.171 smrat::subtropical::Encoding Class Reference

```
#include <Subtropical.h>
```

### Public Member Functions

- [FormulaT encode\\_separator](#) (const [Separator](#) &separator, const [Direction](#) direction, const [SeparatorType](#) separator\_type)
 

*Creates the formula for the hyperplane that linearly separates at least one variant positive frame vertex from all variant negative frame vertices.*
- [FormulaT encode\\_integer](#) (carl::Variable var)
- [Model construct\\_assignment](#) (const [Model](#) &lra\_model, const [FormulaT](#) &f) const

### 0.15.171.1 Member Function Documentation

**0.15.171.1.1 construct\_assignment()** [Model](#) smrat::subtropical::Encoding::construct\_assignment (const [Model](#) & lra\_model, const [FormulaT](#) & f ) const [inline]

Stores all informations retrieved from the LRA solver to construct the model

**0.15.171.1.2 encode\_integer()** [FormulaT](#) smrat::subtropical::Encoding::encode\_integer (carl::Variable var ) [inline]

**0.15.171.1.3 encode\_separator()** [FormulaT](#) smrat::subtropical::Encoding::encode\_separator (const [Separator](#) & separator, const [Direction](#) direction, const [SeparatorType](#) separator\_type ) [inline]

Creates the formula for the hyperplane that linearly separates at least one variant positive frame vertex from all variant negative frame vertices.

If a weak separator is searched, the corresponding rating is included.

### Parameters

|                  |                                                                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <i>separator</i> | The separator object that stores the construction information.                                                                                |
| <i>negated</i>   | True, if the formula for the negated polynomial shall be constructed. False, if the formula for the original polynomial shall be constructed. |

**Returns**

Formula that is satisfiable iff such a separating hyperplane exists.

Create the hyperplane and sign change formula

Create the rating case distinction formula

Create the strict/weak linear separating hyperplane

## 0.15.172 smtrat::EQGraph< VertexName > Struct Template Reference

```
#include <EQGraph.h>
```

### Public Types

- using `vec` = boost::vecS
- using `undirected` = boost::undirectedS
- using `vertex_property` = boost::property< vertex\_properties\_t, VertexName >
- using `Graph` = boost::adjacency\_list< `vec`, `vec`, `undirected`, `vertex_property` >
- using `Vertex` = typename boost::graph\_traits< `Graph` >::vertex\_descriptor
- using `PathEdge` = std::pair< VertexName, VertexName >
- using `MaybePath` = std::optional< std::vector< `PathEdge` > >

### Public Member Functions

- auto `properties` () noexcept
- auto `properties` () const noexcept
- VertexName const & `properties` (Vertex const &v) const noexcept
- VertexName & `properties` (Vertex const &v) noexcept
- void `add_vertex` (VertexName const &name) noexcept
- void `add_edge` (VertexName const &a, VertexName const &b) noexcept
- void `remove_edge` (VertexName const &a, VertexName const &b) noexcept
- `MaybePath get_path` (VertexName const &a, VertexName const &b) noexcept
- `MaybePath get_path` (VertexName const &a, VertexName const &b, size\_t bound) noexcept

### Data Fields

- `Graph graph`
- std::map< VertexName, `Vertex` > `vertices`

### 0.15.172.1 Member Typedef Documentation

**0.15.172.1.1 Graph** template<typename VertexName >  
using smtrat::EQGraph< VertexName >::`Graph` = boost::adjacency\_list< `vec`, `vec`, `undirected`,  
`vertex_property`>

**0.15.172.1.2 MaybePath** template<typename VertexName >  
using smtrat::EQGraph< VertexName >::`MaybePath` = std::optional< std::vector< `PathEdge` > >

**0.15.172.1.3 PathEdge** template<typename VertexName >  
using smtrat::EQGraph< VertexName >::`PathEdge` = std::pair< VertexName, VertexName >

**0.15.172.1.4 undirected** template<typename VertexName >  
using smtrat::EQGraph< VertexName >::`undirected` = boost::undirectedS

```
0.15.172.1.5 vec template<typename VertexName >
using smtrat::EQGraph< VertexName >::vec = boost::vecS
```

```
0.15.172.1.6 Vertex template<typename VertexName >
using smtrat::EQGraph< VertexName >::Vertex = typename boost::graph_traits<Graph>::vertex_descriptor
```

```
0.15.172.1.7 vertex_property template<typename VertexName >
using smtrat::EQGraph< VertexName >::vertex_property = boost::property<vertex_properties_t,
VertexName>
```

## 0.15.172.2 Member Function Documentation

```
0.15.172.2.1 add_edge() template<typename VertexName >
void smtrat::EQGraph< VertexName >::add_edge (
 VertexName const & a,
 VertexName const & b) [inline], [noexcept]
```

```
0.15.172.2.2 add_vertex() template<typename VertexName >
void smtrat::EQGraph< VertexName >::add_vertex (
 VertexName const & name) [inline], [noexcept]
```

```
0.15.172.2.3 get_path() [1/2] template<typename VertexName >
MaybePath smtrat::EQGraph< VertexName >::get_path (
 VertexName const & a,
 VertexName const & b) [inline], [noexcept]
```

```
0.15.172.2.4 get_path() [2/2] template<typename VertexName >
MaybePath smtrat::EQGraph< VertexName >::get_path (
 VertexName const & a,
 VertexName const & b,
 size_t bound) [inline], [noexcept]
```

```
0.15.172.2.5 properties() [1/4] template<typename VertexName >
auto smtrat::EQGraph< VertexName >::properties () const [inline], [noexcept]
```

```
0.15.172.2.6 properties() [2/4] template<typename VertexName >
auto smtrat::EQGraph< VertexName >::properties () [inline], [noexcept]
```

```
0.15.172.2.7 properties() [3/4] template<typename VertexName >
VertexName const& smtrat::EQGraph< VertexName >::properties (
 Vertex const & v) const [inline], [noexcept]
```

```
0.15.172.2.8 properties() [4/4] template<typename VertexName >
VertexName& smtrat::EQGraph< VertexName >::properties (
 Vertex const & v) [inline], [noexcept]
```

```
0.15.172.2.9 remove_edge() template<typename VertexName >
void smtrat::EQGraph< VertexName >::remove_edge (
 VertexName const & a,
 VertexName const & b) [inline], [noexcept]
```

### 0.15.172.3 Field Documentation

```
0.15.172.3.1 graph template<typename VertexName >
Graph smtrat::EQGraph< VertexName >::graph
```

```
0.15.172.3.2 vertices template<typename VertexName >
std::map<VertexName, Vertex> smtrat::EQGraph< VertexName >::vertices
```

## 0.15.173 smtrat::EQModule< Settings > Class Template Reference

```
#include <EQModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
- `EQModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~EQModule ()`
- `bool informCore (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `bool addCore (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `void removeCore (ModuleInput::const_iterator _subformula)`  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- `void updateModel () const`  
*Updates the current assignment into the model.*
- `Answer checkCore ()`  
*Checks the received formula for consistency.*
- `void process_merge_lists ()`  
*parts of the public interface that are not part of the general module interface*
- `const term_type * findRepresentative (const term_type &term)`  
*find a "canonical" representative for some term*
- `std::size_t getClass (const term_type &term)`  
*returns the component-component-level class of some term*
- `bool inform (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*

- void **deinform** (const **FormulaT** &\_constraint)  
*The inverse of informing about a constraint.*
- bool **add** (**ModuleInput::const\_iterator** \_subformula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual **Answer check** (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_←  
VARIABLE)  
*Checks the received formula for consistency.*
- virtual void **remove** (**ModuleInput::const\_iterator** \_subformula)  
*Removes everything related to the given sub-formula of the received formula.*
- virtual void **updateAllModels** ()  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned **currentlySatisfiedByBackend** (const **FormulaT** &\_formula) const
- virtual unsigned **currentlySatisfied** (const **FormulaT** &) const
- bool **receivedVariable** (carl::Variable::Arg \_var) const
- **Answer solverState** () const
- std::size\_t **id** () const
- void **setId** (std::size\_t \_id)  
*Sets this modules unique ID to identify itself.*
- **thread\_priority threadPriority** () const
- void **setThreadPriority** (**thread\_priority** \_threadPriority)  
*Sets the priority of this module to get a thread for running its check procedure.*
- const **ModuleInput** \* **pReceivedFormula** () const
- const **ModuleInput** & **rReceivedFormula** () const
- const **ModuleInput** \* **pPassedFormula** () const
- const **ModuleInput** & **rPassedFormula** () const
- const **Model** & **model** () const
- const std::vector< **Model** > & **allModels** () const
- const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
- const std::vector< **Module** \* > & **usedBackends** () const
- const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
- const carl::FastSet< **FormulaT** > & **informedConstraints** () const
- void **addLemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lt=**LemmaType::NORMAL**, const **FormulaT** &\_preferredFormula=**FormulaT(carl::FormulaType::TRUE)**)  
*Stores a lemma being a valid formula.*
- bool **hasLemmas** ()  
*Checks whether this module or any of its backends provides any lemmas.*
- void **clearLemmas** ()  
*Deletes all yet found lemmas.*
- const std::vector< **Lemma** > & **lemmas** () const
- **ModuleInput::const\_iterator** **firstUncheckedReceivedSubformula** () const
- **ModuleInput::const\_iterator** **firstSubformulaToPass** () const
- void **receivedFormulaChecked** ()  
*Notifies that the received formulas has been checked.*
- const **smrat::Conditionals** & **answerFound** () const
- bool **isPreprocessor** () const
- carl::Variable **objective** () const
- bool **is\_minimizing** () const
- void **excludeNotReceivedVariablesFromModel** () const  
*Excludes all variables from the current model, which do not occur in the received formula.*
- void **updateLemmas** ()

- Stores all lemmas of any backend of this module in its own lemma vector.
- void `collectTheoryPropagations ()`  
Collects the formulas in the given formula, which are part of the received formula.
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.
- void `checkInSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _insubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`
- virtual std::pair< bool, FormulaT > `getReceivedFormulaSimplified ()`
- void `print (const std::string &_initiation="***") const`  
Prints everything relevant of the solver.
- void `printReceivedFormula (const std::string &_initiation="***") const`  
Prints the vector of the received formula.
- void `printPassedFormula (const std::string &_initiation="***") const`  
Prints the vector of passed formula.
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`  
Prints the infeasible subsets.
- void `printModel (std::ostream &_out=std::cout) const`  
Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.
- void `printAllModels (std::ostream &_out=std::cout)`  
Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

### Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
The number of different variables to consider for a probable infinite loop of branchings.
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
Stores the last branches in a cycle buffer.
- static size\_t `mFirstPosInLastBranches` = 0  
The beginning of the cyclic buffer storing the last branches.
- static std::vector< FormulaT > `mOldSplittingVariables`  
Reusable splitting variables.

### Protected Member Functions

- virtual void `deinformCore (const FormulaT &_constraint)`  
The inverse of informing about a constraint.
- bool `anAnswerFound () const`  
Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.
- void `clearModel () const`  
Clears the assignment, if any was found.
- void `clearModels () const`  
Clears all assignments, if any was found.
- void `cleanModel () const`

*Substitutes variable occurrences by its model value in the model values of other variables.*

- `ModuleInput::iterator passedFormulaBegin ()`
- `ModuleInput::iterator passedFormulaEnd ()`
- `void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- `const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`

*Gets the origins of the passed formula at the given position.*
- `void getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
- `void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
- `std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
- `void informBackends (const FormulaT &_constraint)`

*Informs all backends of this module about the given constraint.*
- `virtual void addConstraintToInform (const FormulaT &_constraint)`

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- `std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`

*Copies the given sub-formula of the received formula to the passed formula.*
- `bool originInReceivedFormula (const FormulaT &_origin) const`
- `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)`

*Adds the given formula to the passed formula with no origin.*
- `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`

*Adds the given formula to the passed formula.*
- `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`

*Adds the given formula to the passed formula.*
- `void generateTrivialInfeasibleSubset ()`

*Stores the trivial infeasible subset being the set of received formulas.*
- `void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

*Stores an infeasible subset consisting only of the given received formula.*
- `std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const`
- `void getInfeasibleSubsets ()`

*Copies the infeasible subsets of the passed formula.*
- `std::vector< FormulaSetT > getInfeasibleSubsets (const Model & backend) const`

*Get the infeasible subsets the given backend provides.*
- `const Model & backendsModel () const`

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- `void getBackendsModel () const`
- `void getBackendsAllModels () const`

*Stores all models of a backend in the list of all models of this module.*
- `virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*
- `virtual Answer runBackends ()`
- `virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- `void clearPassedFormula ()`
- `std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`

*Merges the two vectors of sets into the first one.*

- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename `Poly::PolyType` &\_branchingPolynomial, const `Rational` &\_← branchingValue) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &← \_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
 

*Adds a formula to the `InformationRelevantFormula`.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
 

*Gets all `InformationRelevantFormulas`.*
- bool `isLemmaLevel` (`LemmaLevel` level)
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
 

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- `std::vector< Module * > mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- `std::vector< Module * > mAllBackends`  
*The backends of this module which have been used.*
- `std::vector< Lemma > mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- `carl::FastSet< FormulaT > mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.173.1 Member Typedef Documentation

**0.15.173.1.1 SettingsType** `template<typename Settings >`  
`typedef Settings smtrat::EQModule< Settings >::SettingsType`

### 0.15.173.2 Member Enumeration Documentation

**0.15.173.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.173.3 Constructor & Destructor Documentation

**0.15.173.3.1 EQModule()** `template<typename Settings >`  
`smtrat::EQModule< Settings >::EQModule (`  
 `const ModuleInput * _formula,`  
 `Conditionals & _conditionals,`  
 `Manager * _manager = NULL )`

**0.15.173.3.2 ~EQModule()** template<typename Settings >  
smtrat::EQModule< Settings >::~EQModule ( )

#### 0.15.173.4 Member Function Documentation

**0.15.173.4.1 add()** bool smtrat::Module::add (   
    ModuleInput::const\_iterator \_subformula ) [inherited]

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.173.4.2 addConstraintToInform()** void smtrat::Module::addConstraintToInform (   
    const FormulaT & \_constraint ) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

##### Parameters

|             |                        |
|-------------|------------------------|
| _constraint | The constraint to add. |
|-------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.173.4.3 addCore()** template<typename Settings >  
bool smtrat::EQModule< Settings >::addCore (   
    ModuleInput::const\_iterator \_subformula ) [virtual]

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.173.4.4 addInformationRelevantFormula()** void smtrat::Module::addInformationRelevantFormula (   
    const FormulaT & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

##### Parameters

|         |                |
|---------|----------------|
| formula | Formula to add |
|---------|----------------|

```
0.15.173.4.5 addLemma() void smtrat::Module::addLemma (
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

```
0.15.173.4.6 addOrigin() void smtrat::Module::addOrigin (
```

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

```
0.15.173.4.7 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>_subformula</i> | The sub-formula of the received formula to copy. |
|--------------------|--------------------------------------------------|

```
0.15.173.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat<-
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula. |
|-----------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.173.4.9 addSubformulaToPassedFormula()** [2/3] std::pair<ModuleInput::iterator,bool> smrat::Module::addSubformulaToPassedFormula (

```
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                 |
| <i>_origin</i>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.173.4.10 addSubformulaToPassedFormula()** [3/3] std::pair<ModuleInput::iterator,bool> smrat::Module::addSubformulaToPassedFormula (

```
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.173.4.11 allModels()** const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]

**Returns**

All satisfying assignments, if existent.

**0.15.173.4.12 `anAnswerFound()`** `bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.173.4.13 `answerFound()`** `const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]`

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.173.4.14 `backendsModel()`** `const Model & smrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.173.4.15 `branchAt() [1/4]`** `bool smrat::Module::branchAt ( carl::Variable::Arg _var, const Rational & _value, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]`

**0.15.173.4.16 `branchAt() [2/4]`** `bool smrat::Module::branchAt ( carl::Variable::Arg _var, const Rational & _value, std::vector< FormulaT > && _premise, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]`

**0.15.173.4.17 `branchAt() [3/4]`** `bool smrat::Module::branchAt ( const Poly & _polynomial, bool _integral, const Rational & _value, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]`

**0.15.173.4.18 `branchAt() [4/4]`** `bool smrat::Module::branchAt ( const Poly & _polynomial, bool _integral, const Rational & _value,`

```
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                              |                                                                                                                                                                                                                |
|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">_polynomial</a>                  | The variable to branch for.                                                                                                                                                                                    |
| <a href="#">_integral</a>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <a href="#">_value</a>                       | The value to branch at.                                                                                                                                                                                        |
| <a href="#">_premise</a>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <a href="#">_leftCaseWeak</a>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <a href="#">_preferLeftCase</a>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <a href="#">_useReceivedFormulaAsPremise</a> | true, if the whole received formula should be used as premise                                                                                                                                                  |

#### 0.15.173.4.19 [check\(\)](#) Answer [smtrat::Module](#)::check (

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                            |                                                                                                                                       |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">_final</a>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <a href="#">_full</a>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <a href="#">_objective</a> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

#### 0.15.173.4.20 [checkCore\(\)](#) template<typename Settings >

```
Answer smtrat::EQModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.173.4.21 checkInfSubsetForMinimality() void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.173.4.22 checkModel() unsigned smtrat::Module::checkModel () const [protected], [inherited]
```

#### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

```
0.15.173.4.23 cleanModel() void smtrat::Module::cleanModel () const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

```
0.15.173.4.24 clearLemmas() void smtrat::Module::clearLemmas () [inline], [inherited]
```

Deletes all yet found lemmas.

```
0.15.173.4.25 clearModel() void smtrat::Module::clearModel () const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

```
0.15.173.4.26 clearModels() void smtrat::Module::clearModels () const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

```
0.15.173.4.27 clearPassedFormula() void smtrat::Module::clearPassedFormula () [protected], [inherited]
```

```
0.15.173.4.28 collectOrigins() [1/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

```
0.15.173.4.29 collectOrigins() [2/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

```
0.15.173.4.30 collectTheoryPropagations() void smtrat::Module::collectTheoryPropagations () [inherited]
```

```
0.15.173.4.31 constraintsToInform() const carl::FastSet<FormulaT>& smtrat::Module::constraints←ToInform () const [inline], [inherited]
```

#### Returns

The constraints which the backends must be informed about.

```
0.15.173.4.32 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAbstractModule< Settings >](#), [smtrat::LRAbstractModule< LRASettingsICP >](#), and [smtrat::LRAbstractModule< LRASettings >](#)

```
0.15.173.4.33 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.173.4.34 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.173.4.35 `deinformCore()`** `virtual void smtrat::Module::deinformCore (`  
 `const FormulaT & _constraint )` [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAbstractModule< Settings >](#), [smtrat::LRAbstractModule< LRASettingsICP >](#), and [smtrat::LRAbstractModule< LRASettings1 >](#).

**0.15.173.4.36 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (`  
 `const std::vector< FormulaT > & origins ) const` [protected], [inherited]

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.173.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`  
 `ModuleInput::iterator _subformula,`  
 `bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.173.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.173.4.39 `find_representative()`** `template<typename Settings >`  
`const term_type* smtrat::EQModule< Settings >::find_representative (`  
 `const term_type & term )`  
find a "canonical" representative for some term

```
0.15.173.4.40 findBestOrigin() std::vector< FormulaT >::const_iterator smrat::Module::findBestOrigin (const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

**Parameters**

|                 |
|-----------------|
| <i>_origins</i> |
|-----------------|

**Returns**

```
0.15.173.4.41 firstSubformulaToPass() ModuleInput::const_iterator smrat::Module::firstSubformulaToPass () const [inline], [inherited]
```

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

```
0.15.173.4.42 firstUncheckedReceivedSubformula() ModuleInput::const_iterator smrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]
```

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

```
0.15.173.4.43 freeSplittingVariable() static void smrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable) [inline], [static], [inherited]
```

```
0.15.173.4.44 generateTrivialInfeasibleSubset() void smrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

```
0.15.173.4.45 get_class() template<typename Settings > std::size_t smrat::EQModule< Settings >::get_class (const term_type & term)
```

returns the component-component-level class of some term

```
0.15.173.4.46 getBackendsAllModels() void smrat::Module::getBackendsAllModels () const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

```
0.15.173.4.47 getBackendsModel() void smrat::Module::getBackendsModel () const [protected], [inherited]
```

**0.15.173.4.48 getInfeasibleSubsets()** [1/2] void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.173.4.49 getInfeasibleSubsets()** [2/2] std::vector< [FormulaSetT](#) > smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.173.4.50 getInformationRelevantFormulas()** const std::set< [FormulaT](#) > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.173.4.51 getModelEqualities()** std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.173.4.52 getOrigins()** [1/3] void smtrat::Module::getOrigins ( ) const [inline], [protected], [inherited]  
const [FormulaT](#) & `_formula`,  
[FormulaSetT](#) & `_origins` ) const [inline], [protected], [inherited]

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.173.4.53 getOrigins()** [2/3] void smtrat::Module::getOrigins ( ) const [inline], [protected], [inherited]  
const [FormulaT](#) & `_formula`,  
[FormulasT](#) & `_origins` ) const [inline], [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_formula</i> |  |
| <i>_origins</i> |  |

**0.15.173.4.54 `getOrigins()` [3/3]** const `FormulaT&` `smtrat::Module::getOrigins (ModuleInput::const_iterator _formula )` const [inline], [protected], [inherited]  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.173.4.55 `getReceivedFormulaSimplified()`** std::pair< bool, `FormulaT` > `smtrat::Module::get←ReceivedFormulaSimplified ( )` [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.173.4.56 `hasLemmas()`** bool `smtrat::Module::hasLemmas ( )` [inline], [inherited]  
Checks whether this module or any of its backends provides any lemmas.

**0.15.173.4.57 `hasValidInfeasibleSubset()`** bool `smtrat::Module::hasValidInfeasibleSubset ( )` const [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.173.4.58 `id()`** std::size\_t `smtrat::Module::id ( )` const [inline], [inherited]

**Returns**

A unique ID to identify this module instance.

**0.15.173.4.59 `infeasibleSubsets()`** const std::vector<`FormulaSetT`>& `smtrat::Module::infeasible←Subsets ( )` const [inline], [inherited]

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

```
0.15.173.4.60 inform() bool smrat::Module::inform (
 const FormulaT & _constraint) [inherited]
```

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

#### Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

```
0.15.173.4.61 informBackends() void smrat::Module::informBackends (
```

```
 const FormulaT & _constraint) [inline], [protected], [inherited]
```

Informs all backends of this module about the given constraint.

#### Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

```
0.15.173.4.62 informCore() template<typename Settings >
```

```
bool smrat::EQModule< Settings >::informCore (
```

```
 const FormulaT & _constraint) [virtual]
```

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

#### Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smrat::Module](#).

```
0.15.173.4.63 informedConstraints() const carl::FastSet<FormulaT>& smrat::Module::informedConstraints () const [inline], [inherited]
```

#### Returns

The position of the first constraint of which no backend has been informed about.

```
0.15.173.4.64 init() template<typename Settings >
```

```
void smrat::EQModule< Settings >::init () [virtual]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not and will not be called more than once and only before the first runBackends call.

Reimplemented from [smrat::Module](#).

**0.15.173.4.65 `is_minimizing()`** `bool smtrat::Module::is_minimizing () const [inline], [inherited]`

**0.15.173.4.66 `isLemmaLevel()`** `bool smtrat::Module::isLemmaLevel (`

`LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.173.4.67 `isPreprocessor()`** `bool smtrat::Module::isPreprocessor () const [inline], [inherited]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.173.4.68 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas () const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.173.4.69 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (`

`const std::vector< FormulaT > & _vecSetA,`

`const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

Returns

The vector being the two given vectors merged.

**0.15.173.4.70 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

**0.15.173.4.71 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (`

`const Model & _modelA,`

`const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.173.4.72 `moduleName()`** `template<typename Settings >`  
`std::string smtrat::EQModule< Settings >::moduleName ( ) const [inline], [virtual]`

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.173.4.73 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.173.4.74 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`  
`const FormulaT & _origin ) const [protected], [inherited]`

**0.15.173.4.75 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin (`  
`) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.173.4.76 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
`[inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.173.4.77 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.173.4.78 `pReceivedFormula()`** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.173.4.79 `print()`** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.173.4.80 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.173.4.81 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.173.4.82 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.173.4.83 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.173.4.84 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.173.4.85 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                      |                                                                                     |
|----------------------|-------------------------------------------------------------------------------------|
| _branchingPolynomial | The polynomial to branch at (in most branching strategies this is just a variable). |
| _branchingValue      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.173.4.86 process_merge_lists() template<typename Settings >
void smtrat::EQModule< Settings >::process_merge_lists ()
parts of the public interface that are not part of the general module interface
```

```
0.15.173.4.87 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline],
[inherited]
```

Notifies that the received formulas has been checked.

```
0.15.173.4.88 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs<=
InfeasibleSubset (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.173.4.89 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.173.4.90 remove() void smtrat::Module::remove (
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub formula of the received formula to remove. |
|-------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

```
0.15.173.4.91 removeCore() template<typename Settings >
void smtrat::EQModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.173.4.92 `removeOrigin()`** `std::pair<ModuleInput::iterator, bool> smrat::Module::removeOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.173.4.93 `removeOrigins()`** `std::pair<ModuleInput::iterator, bool> smrat::Module::removeOrigins (`

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.173.4.94 `rPassedFormula()`** `const ModuleInput& smrat::Module::rPassedFormula ( ) const`  
[inline], [inherited]

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.173.4.95 `rReceivedFormula()`** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const`  
[inline], [inherited]

#### Returns

A reference to the formula passed to this module.

**0.15.173.4.96 `runBackends() [1/2]`** `virtual Answer smrat::Module::runBackends ( ) [inline],`  
[protected], [virtual], [inherited]

Reimplemented in [smrat::PModule](#).

**0.15.173.4.97 `runBackends() [2/2]`** `Answer smrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.173.4.98 setId()** `void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`  
 Sets this modules unique ID to identify itself.

**Parameters**

|              |                                    |
|--------------|------------------------------------|
| $\leftarrow$ | The id to set this module's id to. |
| $\_id$       |                                    |

**0.15.173.4.99 setThreadPriority()** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`  
 Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| $\_threadPriority$ | The priority to set this module's thread priority to. |
|--------------------|-------------------------------------------------------|

**0.15.173.4.100 solverState()** [Answer](#) `smtrat::Module::solverState ( ) const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.173.4.101 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`  
 Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                       |                                                  |
|-----------------------|--------------------------------------------------|
| $\_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|-----------------------|--------------------------------------------------|

**0.15.173.4.102 threadPriority()** [thread\\_priority](#) `smtrat::Module::threadPriority ( ) const [inline], [inherited]`

**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.173.4.103 updateAllModels()** void smtrat::Module::updateAllModels () [virtual], [inherited]  
Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.173.4.104 updateLemmas()** void smtrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.173.4.105 updateModel()** template<typename Settings >  
void [smtrat::EQModule< Settings >](#)::updateModel () const [virtual]  
Updates the current assignment into the model.  
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.173.4.106 usedBackends()** const std::vector<[Module](#)\*>& smtrat::Module::usedBackends ()  
const [inline], [inherited]

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.173.5 Field Documentation

**0.15.173.5.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected],  
[inherited]  
The backends of this module which have been used.

**0.15.173.5.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected],  
[inherited]  
Stores all satisfying assignments.

**0.15.173.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer  
[protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.173.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform  
[protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.173.5.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.173.5.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.173.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass` [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.173.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.173.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer` [protected], [inherited]

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.173.5.10 mFullCheck** `bool smrat::Module::mFullCheck` [protected], [inherited]

false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.173.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smrat::Module::mInfeasibleSubsets`

[protected], [inherited]

Stores the infeasible subsets.

**0.15.173.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints`

[protected], [inherited]

Stores the position of the first constraint of which no backend has been informed about.

**0.15.173.5.13 mLastBranches** `std::vector< Branching > smrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.173.5.14 mLemmas** `std::vector<Lemma> smrat::Module::mLemmas` [protected], [inherited]

Stores the lemmas being valid formulas this module or its backends made.

**0.15.173.5.15 mModel** `Model smrat::Module::mModel` [mutable], [protected], [inherited]

Stores the assignment of the current satisfiable result, if existent.

**0.15.173.5.16 mModelComputed** `bool smrat::Module::mModelComputed` [mutable], [protected], [inherited]

True, if the model has already been computed.

**0.15.173.5.17 mNumOfBranchVarsToStore** `std::size_t smrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.173.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.173.5.19 mOldSplittingVariables** `std::vector< FormulaT > smtrat::Module::mOldSplitting [static], [inherited]`  
Reusable splitting variables.

**0.15.173.5.20 mpManager** `Manager* const smtrat::Module::mpManager [protected], [inherited]`  
A reference to the manager.

**0.15.173.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.173.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.173.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

**0.15.173.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.173.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`  
Maps variables to the number of their occurrences.

## 0.15.174 smtrat::EQPreprocessingModule< Settings > Class Template Reference

#include <EQPreprocessingModule.h>

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
- `EQPreprocessingModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~EQPreprocessingModule ()`
- `void updateModel () const`  
*Updates the model, if the solver has detected the consistency of the received formula, beforehand.*
- `Answer checkCore ()`

- Checks the received formula for consistency.
- bool `inform` (const `FormulaT` &\_constraint)
  - Informs the module about the given constraint.
- void `deinform` (const `FormulaT` &\_constraint)
  - The inverse of informing about a constraint.
- virtual void `init` ()
  - Informs all backends about the so far encountered constraints, which have not yet been communicated.
- bool `add` (`ModuleInput::const_iterator` \_subformula)
  - The module has to take the given sub-formula of the received formula into account.
- virtual `Answer check` (bool \_final=false, bool \_full=true, `carl::Variable` \_objective=`carl::Variable::NO_VARIABLE`)
  - Checks the received formula for consistency.
- virtual void `remove` (`ModuleInput::const_iterator` \_subformula)
  - Removes everything related to the given sub-formula of the received formula.
- virtual void `updateAllModels` ()
  - Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- virtual `std::list< std::vector< carl::Variable > > getModelEqualities` () const
  - Partition the variables from the current model into equivalence classes according to their assigned value.
- unsigned `currentlySatisfiedByBackend` (const `FormulaT` &\_formula) const
- virtual unsigned `currentlySatisfied` (const `FormulaT` &) const
- bool `receivedVariable` (`carl::Variable::Arg` \_var) const
- `Answer solverState` () const
- `std::size_t id` () const
- void `setId` (`std::size_t` \_id)
  - Sets this modules unique ID to identify itself.
- `thread_priority threadPriority` () const
- void `setThreadPriority` (`thread_priority` \_threadPriority)
  - Sets the priority of this module to get a thread for running its check procedure.
- const `ModuleInput * pReceivedFormula` () const
- const `ModuleInput & rReceivedFormula` () const
- const `ModuleInput * pPassedFormula` () const
- const `ModuleInput & rPassedFormula` () const
- const `Model & model` () const
- const `std::vector< Model > & allModels` () const
- const `std::vector< FormulaSetT > & infeasibleSubsets` () const
- const `std::vector< Module * > & usedBackends` () const
- const `carl::FastSet< FormulaT > & constraintsToInform` () const
- const `carl::FastSet< FormulaT > & informedConstraints` () const
- void `addLemma` (const `FormulaT` &\_lemma, const `LemmaType` &\_lt=`LemmaType::NORMAL`, const `FormulaT` &\_preferredFormula=`FormulaT(carl::FormulaType::TRUE)`)
  - Stores a lemma being a valid formula.
- bool `hasLemmas` ()
  - Checks whether this module or any of its backends provides any lemmas.
- void `clearLemmas` ()
  - Deletes all yet found lemmas.
- const `std::vector< Lemma > & lemmas` () const
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula` () const
- `ModuleInput::const_iterator firstSubformulaToPass` () const
- void `receivedFormulaChecked` ()
  - Notifies that the received formulas has been checked.
- const `smrat::Conditionals & answerFound` () const
- bool `isPreprocessor` () const

- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- `void updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfeasibleSubsetForMinimality (std::vector<FormulaSetT>::const_iterator _insubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- `virtual std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- `void printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- `void printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- `void printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- `void printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- `void printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- `static void freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- `static size_t mNumOfBranchVarsToStore = 5`

*The number of different variables to consider for a probable infinite loop of branchings.*
- `static std::vector< Branching > mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`

*Stores the last branches in a cycle buffer.*
- `static size_t mFirstPosInLastBranches = 0`

*The beginning of the cyclic buffer storing the last branches.*
- `static std::vector< FormulaT > mOldSplittingVariables`

*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)
 

*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)
 

*The inverse of informing about a constraint.*
- virtual bool `addCore` (`ModuleInput::const_iterator` formula)
 

*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (`ModuleInput::const_iterator` formula)
 

*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const
 

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const
 

*Clears the assignment, if any was found.*
- void `clearModels` () const
 

*Clears all assignments, if any was found.*
- void `cleanModel` () const
 

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
 

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const
 

*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()

- Copies the infeasible subsets of the passed formula.
  - std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
    - Get the infeasible subsets the given backend provides.
  - const [Model](#) & [backendsModel](#) () const
    - Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
  - void [getBackendsModel](#) () const
  - void [getBackendsAllModels](#) () const
    - Stores all models of a backend in the list of all models of this module.
- virtual [Answer](#) [runBackends](#) (bool \_final, bool \_full, carl::Variable \_objective)
  - Runs the backend solvers on the passed formula.
- virtual [Answer](#) [runBackends](#) ()
- virtual [ModuleInput](#)::iterator [eraseSubformulaFromPassedFormula](#) ([ModuleInput](#)::iterator \_subformula, bool \_ignoreOrigins=false)
  - Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void [clearPassedFormula](#) ()
- std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const
  - Merges the two vectors of sets into the first one.
- size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
- bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const [Rational](#) &\_branchingValue) const
  - Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &&\_premise=std::vector< [FormulaT](#) >())
- bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
- void [splitUnequalConstraint](#) (const [FormulaT](#) &)
  - Adds the following lemmas for the given constraint  $p \neq 0$ .
- unsigned [checkModel](#) () const
- void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)
  - Adds a formula to the InformationRelevantFormula.
- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()
  - Gets all InformationRelevantFormulas.
- bool [isLemmaLevel](#) ([LemmaLevel](#) level)
  - Checks if current lemma level is greater or equal to given level.

## Static Protected Member Functions

- static bool [modelsDisjoint](#) (const [Model](#) &\_modelA, const [Model](#) &\_modelB)
  - Checks whether there is no variable assigned by both models.

## Protected Attributes

- std::vector< [FormulaSetT](#) > mInfeasibleSubsets  
*Stores the infeasible subsets.*
- Manager \*const mpManager  
*A reference to the manager.*
- Model mModel  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< [Model](#) > mAllModels  
*Stores all satisfying assignments.*
- bool mModelComputed  
*True, if the model has already been computed.*
- bool mFinalCheck  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool mFullCheck  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable mObjectiveVariable  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< [TheoryPropagation](#) > mTheoryPropagations
- std::atomic< [Answer](#) > mSolverState  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* mBackendsFoundAnswer  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals mFoundAnswer  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< [Module](#) \* > mUsedBackends  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< [Module](#) \* > mAllBackends  
*The backends of this module which have been used.*
- std::vector< [Lemma](#) > mLemmas  
*Stores the lemmas being valid formulas this module or its backends made.*
- ModuleInput::iterator mFirstSubformulaToPass  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< [FormulaT](#) > mConstraintsToInform  
*Stores the constraints which the backends must be informed about.*
- carl::FastSet< [FormulaT](#) > mInformedConstraints  
*Stores the position of the first constraint of which no backend has been informed about.*
- ModuleInput::const\_iterator mFirstUncheckedReceivedSubformula  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned mSmallerMusesCheckCounter  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > mVariableCounters  
*Maps variables to the number of their occurrences.*

### 0.15.174.1 Member Typedef Documentation

```
0.15.174.1.1 SettingsType template<typename Settings >
typedef Settings smtrat::EQPreprocessingModule< Settings >::SettingsType
```

## 0.15.174.2 Member Enumeration Documentation

### 0.15.174.2.1 LemmaType enum `smtrat::Module::LemmaType` : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

## 0.15.174.3 Constructor & Destructor Documentation

```
0.15.174.3.1 EQPreprocessingModule() template<typename Settings>
smtrat::EQPreprocessingModule<Settings>::EQPreprocessingModule(
 const ModuleInput * _formula,
 Conditionals & _conditionals,
 Manager * _manager = NULL)
```

```
0.15.174.3.2 ~EQPreprocessingModule() template<typename Settings>
smtrat::EQPreprocessingModule<Settings>::~EQPreprocessingModule()
```

## 0.15.174.4 Member Function Documentation

```
0.15.174.4.1 add() bool smtrat::Module::add(
 ModuleInput::const_iterator _subformula) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

```
0.15.174.4.2 addConstraintToInform() void smtrat::Module::addConstraintToInform(
 const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

|             |                        |
|-------------|------------------------|
| _constraint | The constraint to add. |
|-------------|------------------------|

Reimplemented in `smtrat::SATModule<Settings>`.

```
0.15.174.4.3 addCore() virtual bool smtrat::Module::addCore(
```

```
ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

#### 0.15.174.4.4 addInformationRelevantFormula()

```
void smrat::Module::addInformationRelevantFormula(
```

```
 const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

#### 0.15.174.4.5 addLemma()

```
void smrat::Module::addLemma(
```

```
 const FormulaT & _lemma,
```

```
 const LemmaType & _lt = LemmaType::NORMAL,
```

```
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
```

[inherited]

Stores a lemma being a valid formula.

#### Parameters

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

#### 0.15.174.4.6 addOrigin()

```
void smrat::Module::addOrigin(
```

```
 ModuleInput::iterator _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.174.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addReceivedSubformulaToPassedFormula (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.174.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.174.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.174.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.174.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline],
[inherited]
```

#### Returns

All satisfying assignments, if existent.

```
0.15.174.4.12 anAnswerFound() bool smrat::Module::anAnswerFound () const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

```
0.15.174.4.13 answerFound() const smrat::Conditionals& smrat::Module::answerFound () const
[inline], [inherited]
```

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

```
0.15.174.4.14 backendsModel() const Model & smrat::Module::backendsModel () const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.174.4.15** **branchAt()** [1/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.174.4.16** **branchAt()** [2/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.174.4.17** **branchAt()** [3/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.174.4.18** **branchAt()** [4/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.174.4.19 check() Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

```
0.15.174.4.20 checkCore() template<typename Settings >
Answer smtrat::EQPreprocessingModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.174.4.21 checkInfeasibleSubsetForMinimality() void smtrat::Module::checkInfeasibleSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.174.4.22 checkModel() unsigned smtrat::Module::checkModel () const [protected], [inherited]
```

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.174.4.23 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.174.4.24 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.174.4.25 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.174.4.26 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.174.4.27 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.174.4.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.174.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.174.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.174.4.31 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.174.4.32 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.174.4.33 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.174.4.34 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.174.4.35 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.174.4.36 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.174.4.37 eraseSubformulaFromPassedFormula()** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

```
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.174.4.38 excludeNotReceivedVariablesFromModel()** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.174.4.39 findBestOrigin()** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`

```
 const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

**Parameters**

|                       |
|-----------------------|
| <code>_origins</code> |
|-----------------------|

**Returns****0.15.174.4.40 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.174.4.41 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.174.4.42 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.174.4.43 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.174.4.44 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.174.4.45 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.174.4.46 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.174.4.47 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.174.4.48 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.174.4.49 getModelEqualities()** std::list< std::vector< `carl::Variable` > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.174.4.50 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.174.4.51 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.174.4.52 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

**Parameters**

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.174.4.53 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.174.4.54 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.174.4.55 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.174.4.56 `id()`** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.174.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.174.4.58 `inform()`** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.174.4.59 `informBackends()`** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.174.4.60 `informCore()`** `virtual bool smtrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.174.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.174.4.62 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.174.4.63 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.174.4.64 isLemmaLevel()** `bool smrat::Module::isLemmaLevel( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.174.4.65 isPreprocessor()** `bool smrat::Module::isPreprocessor( ) const [inline], [inherited]`

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.174.4.66 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.174.4.67 merge()** `std::vector< FormulaT > smrat::Module::merge( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.174.4.68 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.174.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.174.4.70 `moduleName()`** `template<typename Settings > std::string smtrat::EQPreprocessingModule< Settings >::moduleName () const [inline], [virtual]`**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.174.4.71 `objective()`** `carl::Variable smtrat::Module::objective () const [inline], [inherited]`**0.15.174.4.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (const FormulaT & _origin ) const [protected], [inherited]`**0.15.174.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin () [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.174.4.74 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ()`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.174.4.75 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula () const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.174.4.76 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula () const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.174.4.77 print()** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const` [inherited]  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.174.4.78 printAllModels()** `void smtrat::Module::printAllModels (`  
`std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.174.4.79 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets (`  
`const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.174.4.80 printModel()** `void smtrat::Module::printModel (`  
`std::ostream & _out = std::cout ) const` [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.174.4.81 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.174.4.82 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.174.4.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.174.4.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.174.4.85 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

```
0.15.174.4.86 receivedVariable() bool smrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.174.4.87 remove() void smrat::Module::remove (
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub formula of the received formula to remove. |
|-------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

```
0.15.174.4.88 removeCore() virtual void smrat::Module::removeCore (
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| formula | The sub formula of the received formula to remove. |
|---------|----------------------------------------------------|

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

```
0.15.174.4.89 removeOrigin() std::pair<ModuleInput::iterator,bool> smrat::Module::remove<-
Origin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

```
0.15.174.4.90 removeOrigins() std::pair<ModuleInput::iterator,bool> smrat::Module::remove<-
Origins (
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

```
0.15.174.4.91 rPassedFormula() const ModuleInput& smrat::Module::rPassedFormula () const
[inline], [inherited]
```

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.174.4.92 rReceivedFormula()** `const ModuleInput& smtrat::Module::rReceivedFormula () const [inline], [inherited]`

**Returns**

A reference to the formula passed to this module.

**0.15.174.4.93 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.174.4.94 runBackends() [2/2]** `Answer smtrat::Module::runBackends (`

`bool _final,`

`bool _full,`

`carl::Variable _objective ) [protected], [virtual], [inherited]`

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.174.4.95 setId()** `void smtrat::Module::setId (`

`std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

**Parameters**

|                 |                                    |
|-----------------|------------------------------------|
| <code>id</code> | The id to set this module's id to. |
|-----------------|------------------------------------|

**0.15.174.4.96 setThreadPriority()** `void smtrat::Module::setThreadPriority (`

`thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.174.4.97 `solverState()`** `Answer smtrat::Module::solverState () const [inline], [inherited]`**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.174.4.98 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint (const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.174.4.99 `threadPriority()`** `thread_priority smtrat::Module::threadPriority () const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.174.4.100 `updateAllModels()`** `void smtrat::Module::updateAllModels () [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.174.4.101 `updateLemmas()`** `void smtrat::Module::updateLemmas () [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.174.4.102 `updateModel()`** `template<typename Settings > void smtrat::EQPreprocessingModule< Settings >::updateModel () const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from [smtrat::Module](#).

**0.15.174.4.103 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends () const [inline], [inherited]`**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.174.5 Field Documentation**

**0.15.174.5.1 mAllBackends** `std::vector<Module*> smrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.174.5.2 mAllModels** `std::vector<Model> smrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

**0.15.174.5.3 mBackendsFoundAnswer** `std::atomic_bool* smrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.174.5.4 mConstraintsToInform** `carl::FastSet<FormulaT> smrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

**0.15.174.5.5 mFinalCheck** `bool smrat::Module::mFinalCheck [protected], [inherited]`  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.174.5.6 mFirstPosInLastBranches** `std::size_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

**0.15.174.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.174.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.174.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.174.5.10 mFullCheck** `bool smrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.174.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smrat::Module::mInfeasibleSubsets [protected], [inherited]`

Stores the infeasible subsets.

**0.15.174.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.174.5.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.174.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.174.5.15 mModel** `Model smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.174.5.16 mModelComputed** `bool smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.174.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.174.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.174.5.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.174.5.20 mpManager** `Manager* const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.174.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter` [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.174.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.174.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations` [protected], [inherited]

**0.15.174.5.24 mUsedBackends** std::vector<[Module](#)\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.174.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.175 Minisat::Equal< K > Struct Template Reference

```
#include <Map.h>
```

### Public Member Functions

- bool [operator\(\)](#) (const K &k1, const K &k2) const

### 0.15.175.1 Member Function Documentation

**0.15.175.1.1 operator()()** template<class K >  
bool [Minisat::Equal](#)< K >::operator() (  
 const K & k1,  
 const K & k2 ) const [inline]

## 0.15.176 delta::ErrorHandler Struct Reference

```
#include <Parser.h>
```

### Data Structures

- struct [result](#)

### Public Member Functions

- template<typename T >  
qi::error\_handler\_result [operator\(\)](#) (T b, T e, T where) const

### 0.15.176.1 Member Function Documentation

**0.15.176.1.1 operator()()** template<typename T >  
qi::error\_handler\_result delta::ErrorHandler::operator() (  
 T b,  
 T e,  
 T where ) const [inline]

## 0.15.177 smtrat::parser::ErrorHandler Struct Reference

```
#include <Script.h>
```

### Data Structures

- struct [result](#)

## Public Member Functions

- template<typename T1 , typename T2 , typename T3 , typename T4 >  
qi::error\_handler\_result **operator()** (T1 b, T2 e, T3 where, T4 const &what) const

### 0.15.177.1 Member Function Documentation

```
0.15.177.1.1 operator() template<typename T1 , typename T2 , typename T3 , typename T4 >
qi::error_handler_result smtrat::parser::ErrorHandler::operator() (
 T1 b,
 T2 e,
 T3 where,
 T4 const & what) const [inline]
```

## 0.15.178 smtrat::ESModule< Settings > Class Template Reference

```
#include <ESModule.h>
```

### Public Types

- typedef **Settings** **SettingsType**
- enum class **LemmaType** : unsigned { **NORMAL** = 0 , **PERMANENT** = 1 }

### Public Member Functions

- ESModule** (const **ModuleInput** \***\_formula**, **Conditionals** &**\_conditionals**, **Manager** \***\_manager**=NULL)
- ~ESModule** ()
- void **updateModel** () const  
*Updates the current assignment into the model.*
- Answer checkCore** ()  
*Checks the received formula for consistency.*
- bool **isPreprocessor** () const
- bool **appliedPreprocessing** () const
- bool **add** (**ModuleInput**::const\_iterator **\_subformula**)  
*The module has to take the given sub-formula of the received formula into account.*
- void **remove** (**ModuleInput**::const\_iterator **\_subformula**)  
*Removes everything related to the given sub-formula of the received formula.*
- Answer check** (bool **\_final**=false, bool **\_full**=true, carl::Variable **\_objective**=carl::Variable::NO\_VARIABLE)  
*Checks the received formula for consistency.*
- Answer runBackends** (bool **\_final**, bool **\_full**, carl::Variable **\_objective**)  
*Runs the backend solvers on the passed formula.*
- virtual **Answer runBackends** ()
- std::pair< bool, **FormulaT** > **getReceivedFormulaSimplified** ()
- bool **inform** (const **FormulaT** &**\_constraint**)  
*Informs the module about the given constraint.*
- void **deinform** (const **FormulaT** &**\_constraint**)  
*The inverse of informing about a constraint.*
- virtual void **init** ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- virtual void **updateAllModels** ()  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*

- unsigned `currentlySatisfiedByBackend` (const `FormulaT` &\_formula) const
- virtual unsigned `currentlySatisfied` (const `FormulaT` &) const
- bool `receivedVariable` (carl::Variable::Arg \_var) const
- `Answer solverState` () const
- std::size\_t `id` () const
- void `setId` (std::size\_t \_id)  
*Sets this module's unique ID to identify itself.*
- `thread_priority threadPriority` () const
- void `setThreadPriority` (`thread_priority` \_threadPriority)  
*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput` \* `pReceivedFormula` () const
- const `ModuleInput` & `rReceivedFormula` () const
- const `ModuleInput` \* `pPassedFormula` () const
- const `ModuleInput` & `rPassedFormula` () const
- const `Model` & `model` () const
- const std::vector< `Model` > & `allModels` () const
- const std::vector< `FormulaSetT` > & `infeasibleSubsets` () const
- const std::vector< `Module` \* > & `usedBackends` () const
- const carl::FastSet< `FormulaT` > & `constraintsToInform` () const
- const carl::FastSet< `FormulaT` > & `informedConstraints` () const
- void `addLemma` (const `FormulaT` &\_lemma, const `LemmaType` &\_lt=`LemmaType::NORMAL`, const `FormulaT` &\_preferredFormula=`FormulaT(carl::FormulaType::TRUE)`)  
*Stores a lemma being a valid formula.*
- bool `hasLemmas` ()  
*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas` ()  
*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas` () const
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula` () const
- `ModuleInput::const_iterator firstSubformulaToPass` () const
- void `receivedFormulaChecked` ()  
*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals` & `answerFound` () const
- virtual std::string `moduleName` () const
- carl::Variable `objective` () const
- bool `is_minimizing` () const
- void `excludeNotReceivedVariablesFromModel` () const  
*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas` ()  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations` ()
- void `collectOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const  
*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- bool `isValidInfeasibleSubset` () const
- void `checkInfSubsetForMinimality` (std::vector< `FormulaSetT` >::const\_iterator \_infsSubset, const std::string &\_filename="smaller\_muses", unsigned \_maxSizeDifference=1) const  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print` (const std::string &\_initiation="\*\*\*") const  
*Prints everything relevant of the solver.*
- void `printReceivedFormula` (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of the received formula.*

- void `printPassedFormula` (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const  
*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< `Branching` > `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< `FormulaT` > `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)  
*The inverse of informing about a constraint.*
- virtual bool `addCore` (`ModuleInput`::const\_iterator formula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (`ModuleInput`::const\_iterator formula)  
*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput`::iterator `passedFormulaBegin` ()
- `ModuleInput`::iterator `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput`::iterator \_formula, const `FormulaT` &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput`::const\_iterator \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const

- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
  - std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
    - std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
  - void `informBackends` (const `FormulaT` &\_constraint)
    - Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
  - Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
  - Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
    - Adds the given formula to the passed formula with no origin.*
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
    - Adds the given formula to the passed formula.*
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
    - Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
  - Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
  - Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
  - void `getInfeasibleSubsets` ()
    - Copies the infeasible subsets of the passed formula.*
  - std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
    - Get the infeasible subsets the given backend provides.*
  - const `Model` & `backendsModel` () const
    - Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - void `getBackendsModel` () const
    - Stores all models of a backend in the list of all models of this module.*
  - void `getBackendsAllModels` () const
    - Stores all models of a backend in the list of all models of this module.*
  - std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const
    - Merges the two vectors of sets into the first one.*
  - size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
    - bool `probablyLooping` (const typename `Poly::PolyType` &\_branchingPolynomial, const `Rational` &\_branchingValue) const
      - Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
    - bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
      - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*

- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector<`FormulaT`> &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set<`FormulaT`> & `getInformationRelevantFormulas` ()
 

*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)
 

*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
 

*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector<`FormulaSetT`> `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector<`Model`> `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector<`TheoryPropagation`> `mTheoryPropagations`
- std::atomic<`Answer`> `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals` `mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector<`Module` \*> `mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector<`Module` \*> `mAllBackends`

*The backends of this module which have been used.*

- `std::vector< Lemma > mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- `carl::FastSet< FormulaT > mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.178.1 Member Typedef Documentation

**0.15.178.1.1 SettingsType** `template<typename Settings >`  
`typedef Settings smrat::ESModule< Settings >::SettingsType`

### 0.15.178.2 Member Enumeration Documentation

**0.15.178.2.1 LemmaType** `enum smrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.178.3 Constructor & Destructor Documentation

**0.15.178.3.1 ESModule()** `template<typename Settings >`  
`smrat::ESModule< Settings >::ESModule (`  
`const ModuleInput * _formula,`  
`Conditionals & _conditionals,`  
`Manager * _manager = NULL )`

**0.15.178.3.2 ~ESModule()** `template<typename Settings >`  
`smrat::ESModule< Settings >::~ESModule ( )`

### 0.15.178.4 Member Function Documentation

**0.15.178.4.1 add()** `bool smrat::PModule::add (`  
`ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.178.4.2 `addConstraintToInform()`** `void smtrat::Module::addConstraintToInform ( const FormulaT & _constraint )` [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.178.4.3 `addCore()`** `virtual bool smtrat::Module::addCore ( ModuleInput::const_iterator formula )` [inline], [protected], [virtual], [inherited]

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICEModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::GBModule< Settings >](#).

**0.15.178.4.4 `addInformationRelevantFormula()`** `void smtrat::Module::addInformationRelevantFormula ( const FormulaT & formula )` [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

```
0.15.178.4.5 addLemma() void smtrat::Module::addLemma (
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| _lemma            | The eduction/lemma to store.                                          |
| _lt               | The type of the lemma.                                                |
| _preferredFormula | Hint for the next decision, which formula should be assigned to true. |

```
0.15.178.4.6 addOrigin() void smtrat::Module::addOrigin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

```
0.15.178.4.7 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

```
0.15.178.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat<-
::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.178.4.9 addSubformulaToPassedFormula()** [2/3] std::pair<ModuleInput::iterator,bool> smrat→  
::Module::addSubformulaToPassedFormula ( const FormulaT & \_formula,  
const FormulaT & \_origin ) [inline], [protected], [inherited]  
Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                 |
| _origin  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.178.4.10 addSubformulaToPassedFormula()** [3/3] std::pair<ModuleInput::iterator,bool>  
smrat::Module::addSubformulaToPassedFormula ( const FormulaT & \_formula,  
const std::shared\_ptr<std::vector<FormulaT>> & \_origins ) [inline], [protected],  
[inherited]  
Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                                         |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                               |
| _origins | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.178.4.11 allModels()** const std::vector<Model>& smrat::Module::allModels () const [inline],  
[inherited]

#### Returns

All satisfying assignments, if existent.

**0.15.178.4.12 anAnswerFound()** bool smrat::Module::anAnswerFound () const [inline], [protected],  
[inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

**0.15.178.4.13 answerFound()** const `smtrat::Conditionals`& `smtrat::Module::answerFound` ( ) const [inline], [inherited]

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.178.4.14 appliedPreprocessing()** bool `smtrat::PModule::appliedPreprocessing` ( ) const [inline], [inherited]

#### Returns

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.178.4.15 backendsModel()** const `Model` & `smtrat::Module::backendsModel` ( ) const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.178.4.16 branchAt() [1/4]** bool `smtrat::Module::branchAt` (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.178.4.17 branchAt() [2/4]** bool `smtrat::Module::branchAt` (

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.178.4.18 branchAt() [3/4]** bool `smtrat::Module::branchAt` (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.178.4.19 branchAt() [4/4]** bool `smtrat::Module::branchAt` (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
```

```

 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]

```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

#### 0.15.178.4.20 `check()`

```

Answer smtrat::PModule::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]

```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

#### 0.15.178.4.21 `checkCore()`

```
template<typename Settings >
Answer smtrat::ESModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.178.4.22 checkInfSubsetForMinimality() void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.178.4.23 checkModel() unsigned smtrat::Module::checkModel () const [protected], [inherited]
```

#### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

```
0.15.178.4.24 cleanModel() void smtrat::Module::cleanModel () const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

```
0.15.178.4.25 clearLemmas() void smtrat::Module::clearLemmas () [inline], [inherited]
```

Deletes all yet found lemmas.

```
0.15.178.4.26 clearModel() void smtrat::Module::clearModel () const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

```
0.15.178.4.27 clearModels() void smtrat::Module::clearModels () const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

```
0.15.178.4.28 clearPassedFormula() void smtrat::Module::clearPassedFormula () [protected], [inherited]
```

```
0.15.178.4.29 collectOrigins() [1/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

```
0.15.178.4.30 collectOrigins() [2/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

```
0.15.178.4.31 collectTheoryPropagations() void smtrat::Module::collectTheoryPropagations () [inherited]
```

```
0.15.178.4.32 constraintsToInform() const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]
```

#### Returns

The constraints which the backends must be informed about.

```
0.15.178.4.33 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

```
0.15.178.4.34 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.178.4.35 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.178.4.36 `deinformCore()`** `virtual void smtrat::Module::deinformCore (`  
 `const FormulaT & _constraint )` [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.178.4.37 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (`  
 `const std::vector< FormulaT > & origins )` const [protected], [inherited]

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.178.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`  
 `ModuleInput::iterator _subformula,`  
 `bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.178.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( )` const [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.178.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`  
 `const std::vector< FormulaT > & _origins )` const [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

**0.15.178.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.178.4.42 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.178.4.43 `freeSplittingVariable()`** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable) [inline], [static], [inherited]`

**0.15.178.4.44 `generateTrivialInfeasibleSubset()`** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.178.4.45 `getBackendsAllModels()`** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.178.4.46 `getBackendsModel()`** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.178.4.47 `getInfeasibleSubsets()` [1/2]** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.178.4.48 `getInfeasibleSubsets()` [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.178.4.49 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.178.4.50 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.178.4.51 `getOrigins()` [1/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.178.4.52 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.178.4.53 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.178.4.54 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT >` `smtrat::PModule::getReceivedFormulaSimplified( )` [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.178.4.55 `hasLemmas()`** `bool smtrat::Module::hasLemmas( )` [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.178.4.56 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const` [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.178.4.57 `id()`** `std::size_t smtrat::Module::id( ) const` [inline], [inherited]

**Returns**

A unique ID to identify this module instance.

**0.15.178.4.58 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const` [inline], [inherited]

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.178.4.59 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint )` [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before `assertSubformula` is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.178.4.60 informBackends()** void smrat::Module::informBackends ( const FormulaT & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**0.15.178.4.61 informCore()** virtual bool smrat::Module::informCore (

const FormulaT & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.178.4.62 informedConstraints()** const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.178.4.63 init()** void smrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

**0.15.178.4.64 is\_minimizing()** bool smrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.178.4.65 `isLemmaLevel()`** `bool smtrat::Module::isLemmaLevel (``LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.178.4.66 `isPreprocessor()`** `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.178.4.67 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.178.4.68 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (``const std::vector< FormulaT > & _vecSetA,``const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.178.4.69 `model()`** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.178.4.70 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (``const Model & _modelA,``const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.178.4.71 `moduleName()`** `virtual std::string smtrat::Module::moduleName( ) const [inline], [virtual], [inherited]`

**Returns**

The name of the given module type as name.

Reimplemented in `smtrat::VSModule< Settings >`, `smtrat::UnionFindModule< Settings >`, `smtrat::UFCegarModule< Settings >`, `smtrat::STropModule< Settings >`, `smtrat::SplitSOSModule< Settings >`, `smtrat::PBPPModule< Settings >`, `smtrat::PBGaussModule< Settings >`, `smtrat::NRAILModule< Settings >`, `smtrat::NewCADModule< Settings >`, `smtrat::LVEModule< Settings >`, `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, `smtrat::LRAModule< LRASettings1 >`, `smtrat::IntEqModule< Settings >`, `smtrat::IntBlastModule< Settings >`, `smtrat::IncWidthModule< Settings >`, `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, `smtrat::ICPModule< ICPSettings1 >`, `smtrat::GBModule< Settings >`, `smtrat::FouMoModule< Settings >`, `smtrat::EQPreprocessingModule< Settings >`, `smtrat::EQModule< Settings >`, `smtrat::CurryModule< Settings >`, `smtrat::CubeLIAModule< Settings >`, `smtrat::CSplitModule< Settings >`, `smtrat::BVMModule< Settings >`, and `smtrat::BEModule< Settings >`.

**0.15.178.4.72 `objective()`** `carl::Variable smtrat::Module::objective( ) const [inline], [inherited]`

**0.15.178.4.73 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.178.4.74 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.178.4.75 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.178.4.76 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.178.4.77 pReceivedFormula()** const `ModuleInput*` smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.178.4.78 print()** void smtrat::Module::print ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.178.4.79 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & `_out` = `std::cout` ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.178.4.80 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.178.4.81 printModel()** void smtrat::Module::printModel ( std::ostream & `_out` = `std::cout` ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.178.4.82 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.178.4.83 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

#### Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.178.4.84 probablyLooping()** bool smtrat::Module::probablyLooping (

const typename Poly::PolyType & *\_branchingPolynomial*,  
const Rational & *\_branchingValue* ) const [protected], [inherited]

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.178.4.85 receivedFormulaChecked()** void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.178.4.86 receivedFormulasAsInfeasibleSubset()** void smtrat::Module::receivedFormulasAsInfeasibleSubset (

ModuleInput::const\_iterator *\_subformula* ) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

**0.15.178.4.87 receivedVariable()** bool smtrat::Module::receivedVariable ( carl::Variable::Arg *\_var* ) const [inline], [inherited]

**0.15.178.4.88 remove()** void smtrat::PModule::remove (

ModuleInput::const\_iterator *\_subformula* ) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub formula of the received formula to remove. |
|--------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.178.4.89 removeCore()** `virtual void smrat::Module::removeCore (`  
    `ModuleInput::const_iterator formula ) [inline], [protected], [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.178.4.90 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin (`  
    `ModuleInput::iterator _formula,`  
    `const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.178.4.91 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins (`  
    `ModuleInput::iterator _formula,`  
    `const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],`  
    `[inherited]`

**0.15.178.4.92 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const`  
    `[inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.178.4.93 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const`  
    `[inline], [inherited]`

#### Returns

A reference to the formula passed to this module.

**0.15.178.4.94 runBackends() [1/2]** `virtual Answer smrat::PModule::runBackends ( ) [inline],`  
    `[virtual], [inherited]`

Reimplemented from [smrat::Module](#).

**0.15.178.4.95 runBackends()** [2/2] [Answer](#) smtrat::PModule::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.178.4.96 setId()** void smtrat::Module::setId (

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

#### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <code>id</code> | The id to set this module's id to. |
|-----------------|------------------------------------|

**0.15.178.4.97 setThreadPriority()** void smtrat::Module::setThreadPriority (

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.178.4.98 solverState()** [Answer](#) smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.178.4.99 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint (

```
 const FormulaT & _unequalConstraint) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                             |
|---------------------------------|---------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol !=. |
|---------------------------------|---------------------------------------------|

**0.15.178.4.100 `threadPriority()`** `thread_priority` `smtrat::Module::threadPriority () const [inline], [inherited]`

**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.178.4.101 `updateAllModels()`** `void smtrat::Module::updateAllModels () [virtual], [inherited]`  
Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in `smtrat::SATModule< Settings >`.

**0.15.178.4.102 `updateLemmas()`** `void smtrat::Module::updateLemmas () [inherited]`  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.178.4.103 `updateModel()`** `template<typename Settings > void smtrat::ESModule< Settings >::updateModel () const [virtual]`  
Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from `smtrat::Module`.

**0.15.178.4.104 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends () const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.178.5 Field Documentation

**0.15.178.5.1 `mAllBackends`** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`  
The backends of this module which have been used.

**0.15.178.5.2 `mAllModels`** `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`  
Stores all satisfying assignments.

**0.15.178.5.3 `mBackendsFoundAnswer`** `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.178.5.4 mConstraintsToInform** `carl::FastSet<FormulaT> smrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.178.5.5 mFinalCheck** `bool smrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.178.5.6 mFirstPosInLastBranches** `std::size_t smrat::Module::mFirstPosInLastBranches = 0` [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.178.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.178.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.178.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.178.5.10 mFullCheck** `bool smrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.178.5.11 mInfeasibleSubsets** `std::vector<FormulaSet> smrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.178.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.178.5.13 mLastBranches** `std::vector<Branching> smrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.178.5.14 mLemmas** `std::vector<Lemma> smrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.178.5.15 mModel** `Model smrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.178.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.178.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.178.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.178.5.19 mOldSplittingVariables** std::vector<FormulaT> smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.178.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.178.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.178.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.178.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.178.5.24 mUsedBackends** std::vector<Module\*> smtrat::Module::mUsedBackends [protected], [inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.178.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.179 smtrat::uf\_impl::estimator< IteratorType, IsRandomAccess > Struct Template Reference

Distance estimation for a range delimited by iterators; if the given iterators are random access, returns the result of std::distance in the estimate\_distance function.

```
#include <union_find.h>
```

### Static Public Member Functions

- static std::size\_t `estimate_distance` (IteratorType begin, IteratorType end) noexcept

#### 0.15.179.1 Detailed Description

```
template<typename IteratorType, bool IsRandomAccess = std::is_convertible< typename std::iterator_traits<IteratorType>::iterator_category, std::random_access_iterator_tag >::value>
struct smtrat::uf_impl::estimator< IteratorType, IsRandomAccess >
```

Distance estimation for a range delimited by iterators; if the given iterators are random access, returns the result of `std::distance` in the `estimate_distance` function.

Otherwise, it simply returns 0.

#### 0.15.179.2 Member Function Documentation

```
0.15.179.2.1 estimate_distance() template<typename IteratorType , bool IsRandomAccess = std::is_convertible< typename std::iterator_traits<IteratorType>::iterator_category, std::random_access_iterator_tag >::value>
static std::size_t smtrat::uf_impl::estimator< IteratorType, IsRandomAccess >::estimate_distance (
 IteratorType begin,
 IteratorType end) [inline], [static], [noexcept]
```

### 0.15.180 smtrat::uf\_impl::estimator< IteratorType, false > Struct Template Reference

```
#include <union_find.h>
```

### Static Public Member Functions

- static constexpr std::size\_t `estimate_distance` (IteratorType, IteratorType) noexcept

#### 0.15.180.1 Member Function Documentation

```
0.15.180.1.1 estimate_distance() template<typename IteratorType >
static constexpr std::size_t smtrat::uf_impl::estimator< IteratorType, false >::estimate_distance (
 IteratorType ,
 IteratorType) [inline], [static], [constexpr], [noexcept]
```

### 0.15.181 smtrat::ExactlyOneCommanderEncoder Class Reference

```
#include <ExactlyOneCommanderEncoder.h>
```

### Public Member Functions

- `ExactlyOneCommanderEncoder ()`
- `Rational encodingSize (const ConstraintT &constraint)`
- `bool canEncode (const ConstraintT &constraint)`
- `std::string name ()`
- `std::optional< FormulaT > encode (const ConstraintT &constraint)`

*Encodes an arbitrary constraint.*

**Data Fields**

- std::size\_t **problem\_size**

**Protected Member Functions**

- std::optional< **FormulaT** > **doEncode** (const **ConstraintT** &constraint)
- **FormulaT generateVarChain** (const std::set< carl::Variable > &vars, carl::FormulaType type)

**0.15.181.1 Constructor & Destructor Documentation**

**0.15.181.1.1 ExactlyOneCommanderEncoder()** smtrat::ExactlyOneCommanderEncoder::ExactlyOneCommanderEncoder ( ) [inline]

**0.15.181.2 Member Function Documentation**

**0.15.181.2.1 canEncode()** bool smtrat::ExactlyOneCommanderEncoder::canEncode ( const **ConstraintT** & constraint ) [virtual]  
Implements [smtrat::PseudoBoolEncoder](#).

**0.15.181.2.2 doEncode()** std::optional< **FormulaT** > smtrat::ExactlyOneCommanderEncoder::doEncode ( const **ConstraintT** & constraint ) [protected], [virtual]  
Implements [smtrat::PseudoBoolEncoder](#).

**0.15.181.2.3 encode()** std::optional< **FormulaT** > smtrat::PseudoBoolEncoder::encode ( const **ConstraintT** & constraint ) [inherited]  
Encodes an arbitrary constraint.

Returns

encoded formula

**0.15.181.2.4 encodingSize()** Rational smtrat::ExactlyOneCommanderEncoder::encodingSize ( const **ConstraintT** & constraint ) [virtual]  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

**0.15.181.2.5 generateVarChain()** **FormulaT** smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]

**0.15.181.2.6 name()** std::string smtrat::ExactlyOneCommanderEncoder::name ( ) [inline], [virtual]  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

**0.15.181.3 Field Documentation**

**0.15.181.3.1 problem\_size** std::size\_t smtrat::PseudoBoolEncoder::problem\_size [inherited]

## 0.15.182 smtrat::execution::ExecutionState Class Reference

```
#include <ExecutionState.h>
```

### Public Member Functions

- `ExecutionState ()`
- `void set_add_assertion_handler (std::function< void(const FormulaT &) > f)`
- `void set_add_soft_assertion_handler (std::function< void(const FormulaT &, Rational, const std::string &) > f)`
- `void set_add_objective_handler (std::function< void(const Poly &, bool) > f)`
- `void set_add_annotated_name_handler (std::function< void(const FormulaT &, const std::string &) > f)`
- `void set_remove_assertion_handler (std::function< void(const FormulaT &) > f)`
- `void set_remove_soft_assertion_handler (std::function< void(const FormulaT &) > f)`
- `void set_remove_objective_handler (std::function< void(const Poly &) > f)`
- `void set_remove_annotated_name_handler (std::function< void(const FormulaT &) > f)`
- `bool is_mode (Mode mode) const`
- `void set_logic (carl::Logic logic)`
- `auto logic () const`
- `void add_assertion (const FormulaT &formula)`
- `bool has_assertion (const FormulaT &formula) const`
- `const auto & assertions () const`
- `void add_soft_assertion (const FormulaT &formula, Rational weight, const std::string &id)`
- `bool has_soft_assertion (const FormulaT &formula) const`
- `void add_objective (const Poly &function, bool minimize)`
- `bool has_objective (const Poly &function)`
- `void push ()`
- `void push (size_t n)`
- `void pop ()`
- `void pop (size_t n)`
- `bool has_annotated_name (const std::string &n)`
- `bool has_annotated_name_formula (const smtrat::FormulaT &f)`
- `void annotate_name (const std::string &name, const smtrat::FormulaT &f)`
- `void enter_sat (const smtrat::Model &model, const ObjectiveValues &objectiveValues)`
- `void enter_unsat ()`
- `const smtrat::Model & model () const`
- `const smtrat::ObjectiveValues & objectiveValues () const`
- `void reset ()`
- `void reset_assertions ()`
- `void reset_answer ()`

### 0.15.182.1 Constructor & Destructor Documentation

#### 0.15.182.1.1 ExecutionState() smtrat::execution::ExecutionState::ExecutionState ( ) [inline]

### 0.15.182.2 Member Function Documentation

#### 0.15.182.2.1 add\_assertion() void smtrat::execution::ExecutionState::add\_assertion ( const FormulaT & formula ) [inline]

```
0.15.182.2.2 add_objective() void smtrat::execution::ExecutionState::add_objective (
 const Poly & function,
 bool minimize) [inline]

0.15.182.2.3 add_soft_assertion() void smtrat::execution::ExecutionState::add_soft_assertion (
 const FormulaT & formula,
 Rational weight,
 const std::string & id) [inline]

0.15.182.2.4 annotate_name() void smtrat::execution::ExecutionState::annotate_name (
 const std::string & name,
 const smtrat::FormulaT & f) [inline]

0.15.182.2.5 assertions() const auto& smtrat::execution::ExecutionState::assertions () const
[inline]

0.15.182.2.6 enter_sat() void smtrat::execution::ExecutionState::enter_sat (
 const smtrat::Model & model,
 const ObjectiveValues & objectiveValues) [inline]

0.15.182.2.7 enter_unsat() void smtrat::execution::ExecutionState::enter_unsat () [inline]

0.15.182.2.8 has_annotated_name() bool smtrat::execution::ExecutionState::has_annotated_name (
 const std::string & n) [inline]

0.15.182.2.9 has_annotated_name_formula() bool smtrat::execution::ExecutionState::has_annotated_name_formula (
 const smtrat::FormulaT & f) [inline]

0.15.182.2.10 has_assertion() bool smtrat::execution::ExecutionState::has_assertion (
 const FormulaT & formula) const [inline]

0.15.182.2.11 has_objective() bool smtrat::execution::ExecutionState::has_objective (
 const Poly & function) [inline]

0.15.182.2.12 has_soft_assertion() bool smtrat::execution::ExecutionState::has_soft_assertion (
 const FormulaT & formula) const [inline]

0.15.182.2.13 is_mode() bool smtrat::execution::ExecutionState::is_mode (
 Mode mode) const [inline]

0.15.182.2.14 logic() auto smtrat::execution::ExecutionState::logic () const [inline]
```

**0.15.182.2.15 `model()`** const `smtrat::Model`& `smtrat::execution::ExecutionState::model` ( ) const [inline]

**0.15.182.2.16 `objectiveValues()`** const `smtrat::ObjectiveValues`& `smtrat::execution::ExecutionState::objectiveValues` ( ) const [inline]

**0.15.182.2.17 `pop() [1/2]`** void `smtrat::execution::ExecutionState::pop` ( ) [inline]

**0.15.182.2.18 `pop() [2/2]`** void `smtrat::execution::ExecutionState::pop` ( size\_t *n* ) [inline]

**0.15.182.2.19 `push() [1/2]`** void `smtrat::execution::ExecutionState::push` ( ) [inline]

**0.15.182.2.20 `push() [2/2]`** void `smtrat::execution::ExecutionState::push` ( size\_t *n* ) [inline]

**0.15.182.2.21 `reset()`** void `smtrat::execution::ExecutionState::reset` ( ) [inline]

**0.15.182.2.22 `reset_answer()`** void `smtrat::execution::ExecutionState::reset_answer` ( ) [inline]

**0.15.182.2.23 `reset_assertions()`** void `smtrat::execution::ExecutionState::reset_assertions` ( ) [inline]

**0.15.182.2.24 `set_add_annotated_name_handler()`** void `smtrat::execution::ExecutionState::set_add_annotated_name_handler` ( std::function< void(const `FormulaT` &, const std::string &) > *f* ) [inline]

**0.15.182.2.25 `set_add_assertion_handler()`** void `smtrat::execution::ExecutionState::set_add_assertion_handler` ( std::function< void(const `FormulaT` &) > *f* ) [inline]

**0.15.182.2.26 `set_add_objective_handler()`** void `smtrat::execution::ExecutionState::set_add_objective_handler` ( std::function< void(const `Poly` &, bool) > *f* ) [inline]

**0.15.182.2.27 `set_add_soft_assertion_handler()`** void `smtrat::execution::ExecutionState::set_add_soft_assertion_handler` ( std::function< void(const `FormulaT` &, `Rational`, const std::string &) > *f* ) [inline]

**0.15.182.2.28 `set_logic()`** void `smtrat::execution::ExecutionState::set_logic` ( carl::Logic *logic* ) [inline]

```
0.15.182.2.29 set_remove_annotated_name_handler() void smtrat::execution::ExecutionState::
::set_remove_annotated_name_handler (std::function< void(const FormulaT &) > f) [inline]
```

```
0.15.182.2.30 set_remove_assertion_handler() void smtrat::execution::ExecutionState::set_<
remove_assertion_handler (std::function< void(const FormulaT &) > f) [inline]
```

```
0.15.182.2.31 set_remove_objective_handler() void smtrat::execution::ExecutionState::set_<
remove_objective_handler (std::function< void(const Poly &) > f) [inline]
```

```
0.15.182.2.32 set_remove_soft_assertion_handler() void smtrat::execution::ExecutionState::set_<
remove_soft_assertion_handler (std::function< void(const FormulaT &) > f) [inline]
```

## 0.15.183 smtrat::Executor< Strategy > Class Template Reference

```
#include <Executor.h>
```

### Public Types

- `typedef types::AttributeValue Value`

### Public Member Functions

- `Executor(Strategy &solver)`
- `~Executor()`
- `void add(const smtrat::FormulaT &f)`
- `void addSoft(const smtrat::FormulaT &f, smtrat::Rational weight, const std::string &id)`
- `void annotateName(const smtrat::FormulaT &f, const std::string &name)`
- `void check()`
- `void declareFun(const carl::Variable &)`
- `void declareSort(const std::string &, const unsigned &)`
- `void defineSort(const std::string &, const std::vector<std::string> &, const carl::Sort &)`
- `void eliminateQuantifiers(const smtrat::qe::QEQuery &)`
- `void exit()`
- `void getAssertions()`
- `void getAllModels()`
- `void getAssignment()`
- `void getModel()`
- `void getProof()`
- `void getObjectives()`
- `void getUnsatCore()`
- `void getValue(const std::vector<carl::Variable> &)`
- `void addObjective(const smtrat::Poly &p, smtrat::parser::OptimizationType ot)`
- `void pop(std::size_t n)`
- `void push(std::size_t n)`
- `void reset()`
- `void resetAssertions()`
- `void setLogic(const carl::Logic &logic)`
- `int getExitCode() const`
- `void addInstruction(std::function<void()> bind)`

- bool `hasInstructions () const`
- void `runInstructions ()`
- void `setArtificialVariables (std::vector< smrat::ModelVariable > &&vars)`
- void `cleanModel (smrat::Model &model) const`
- bool `has_info (const std::string &key) const`
- const auto & `get_info (const std::string &key) const`
- template<typename T >  
  T `option (const std::string &key) const`
- bool `printInstruction () const`
- std::ostream & `diagnostic ()`
- void `diagnostic (const std::string &s)`
- std::ostream & `regular ()`
- void `regular (const std::string &s)`
- OutputWrapper `error ()`
- OutputWrapper `warn ()`
- OutputWrapper `info ()`
- virtual void `echo (const std::string &s)`
- void `getInfo (const std::string &key)`
- void `getOption (const std::string &key)`
- void `setInfo (const Attribute &attr)`
- void `setOption (const Attribute &option)`

## Data Fields

- smrat::Answer `lastAnswer`

## Protected Member Functions

- void `setStream (const std::string &s, std::ostream &os)`

## Protected Attributes

- VariantMap< std::string, Value > `infos`
- VariantMap< std::string, Value > `options`
- std::ostream `mRegular`
- std::ostream `mDiagnostic`
- std::map< std::string, std::ofstream > `streams`

### 0.15.183.1 Member Typedef Documentation

**0.15.183.1.1 Value** `typedef types::AttributeValue smrat::parser::InstructionHandler::Value`  
[inherited]

### 0.15.183.2 Constructor & Destructor Documentation

**0.15.183.2.1 Executor()** `template<typename Strategy >`  
`smrat::Executor< Strategy >::Executor (`  
`Strategy & solver ) [inline]`

**0.15.183.2.2 ~Executor()** `template<typename Strategy >`  
`smrat::Executor< Strategy >::~Executor ( ) [inline]`

### 0.15.183.3 Member Function Documentation

**0.15.183.3.1 add()** template<typename Strategy >  
void smtrat::Executor< Strategy >::add (const smtrat::FormulaT & f) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.2 addInstruction()** void smtrat::parser::InstructionHandler::addInstruction (std::function< void()> bind) [inline], [inherited]

**0.15.183.3.3 addObjective()** template<typename Strategy >  
void smtrat::Executor< Strategy >::addObjective (const smtrat::Poly & p, smtrat::parser::OptimizationType ot) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.4 addSoft()** template<typename Strategy >  
void smtrat::Executor< Strategy >::addSoft (const smtrat::FormulaT & f, smtrat::Rational weight, const std::string & id) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.5 annotateName()** template<typename Strategy >  
void smtrat::Executor< Strategy >::annotateName (const smtrat::FormulaT & f, const std::string & name) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.6 check()** template<typename Strategy >  
void smtrat::Executor< Strategy >::check () [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.7 cleanModel()** void smtrat::parser::InstructionHandler::cleanModel (smtrat::Model & model) const [inline], [inherited]

**0.15.183.3.8 declareFun()** template<typename Strategy >  
void smtrat::Executor< Strategy >::declareFun (const carl::Variable & ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.9 declareSort()** template<typename Strategy >  
void smtrat::Executor< Strategy >::declareSort (const std::string & , const unsigned & ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.10 defineSort()** template<typename Strategy >  
void smtrat::Executor< Strategy >::defineSort (const std::string & , const std::vector< std::string > & , const carl::Sort & ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.11 diagnostic() [1/2]** std::ostream& smtrat::parser::InstructionHandler::diagnostic ( ) [inline], [inherited]

**0.15.183.3.12 diagnostic() [2/2]** void smtrat::parser::InstructionHandler::diagnostic (const std::string & s ) [inline], [inherited]

**0.15.183.3.13 echo()** virtual void smtrat::parser::InstructionHandler::echo (const std::string & s ) [inline], [virtual], [inherited]

**0.15.183.3.14 eliminateQuantifiers()** template<typename Strategy >  
void smtrat::Executor< Strategy >::eliminateQuantifiers (const smtrat::qe::QEQuery & ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.15 error()** OutputWrapper smtrat::parser::InstructionHandler::error ( ) [inline], [inherited]

**0.15.183.3.16 exit()** template<typename Strategy >  
void smtrat::Executor< Strategy >::exit ( ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.17 get\_info()** const auto& smtrat::parser::InstructionHandler::get\_info (const std::string & key ) const [inline], [inherited]

**0.15.183.3.18 getAllModels()** template<typename Strategy >  
void smtrat::Executor< Strategy >::getAllModels ( ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.19 getAssertions()** template<typename Strategy >  
void smtrat::Executor< Strategy >::getAssertions ( ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.20 getAssignment()** template<typename Strategy >  
void smtrat::Executor< Strategy >::getAssignment ( ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.21 `getExitCode()`** template<typename Strategy >  
int smtrat::Executor< Strategy >::getExitCode ( ) const [inline]

**0.15.183.3.22  `getInfo()`** void smtrat::parser::InstructionHandler::getInfo ( const std::string & key ) [inline], [inherited]

**0.15.183.3.23  `getModel()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::getModel ( ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.24  `getObjectives()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::getObjectives ( ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.25  `getOption()`** void smtrat::parser::InstructionHandler::getOption ( const std::string & key ) [inline], [inherited]

**0.15.183.3.26  `getProof()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::getProof ( ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.27  `getUnsatCore()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::getUnsatCore ( ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.28  `getValue()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::getValue ( const std::vector< carl::Variable > & ) [inline], [virtual]  
Implements smtrat::parser::InstructionHandler.

**0.15.183.3.29  `has_info()`** bool smtrat::parser::InstructionHandler::has\_info ( const std::string & key ) const [inline], [inherited]

**0.15.183.3.30  `hasInstructions()`** bool smtrat::parser::InstructionHandler::hasInstructions ( ) const [inline], [inherited]

**0.15.183.3.31  `info()`** OutputWrapper smtrat::parser::InstructionHandler::info ( ) [inline], [inherited]

**0.15.183.3.32  `option()`** template<typename T >  
T smtrat::parser::InstructionHandler::option ( const std::string & key ) const [inline], [inherited]

**0.15.183.3.33 `pop()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::pop ( std::size\_t n ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.34 `printInstruction()`** bool smtrat::parser::InstructionHandler::printInstruction ( ) const [inline], [inherited]

**0.15.183.3.35 `push()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::push ( std::size\_t n ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.36 `regular() [1/2]`** std::ostream& smtrat::parser::InstructionHandler::regular ( ) [inline], [inherited]

**0.15.183.3.37 `regular() [2/2]`** void smtrat::parser::InstructionHandler::regular ( const std::string & s ) [inline], [inherited]

**0.15.183.3.38 `reset()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::reset ( ) [inline], [virtual]  
Reimplemented from [smtrat::parser::InstructionHandler](#).

**0.15.183.3.39 `resetAssertions()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::resetAssertions ( ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.40 `runInstructions()`** void smtrat::parser::InstructionHandler::runInstructions ( ) [inline], [inherited]

**0.15.183.3.41 `setArtificialVariables()`** void smtrat::parser::InstructionHandler::setArtificialVariables ( std::vector< smtrat::ModelVariable > && vars ) [inline], [inherited]

**0.15.183.3.42 `setInfo()`** void smtrat::parser::InstructionHandler::setInfo ( const Attribute & attr ) [inline], [inherited]

**0.15.183.3.43 `setLogic()`** template<typename Strategy >  
void smtrat::Executor< Strategy >::setLogic ( const carl::Logic & logic ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.183.3.44 `setOption()`** void smtrat::parser::InstructionHandler::setOption ( const Attribute & option ) [inline], [inherited]

**0.15.183.3.45 `setStream()`** void smtrat::parser::InstructionHandler::setStream ( const std::string & *s*, std::ostream & *os* ) [inline], [protected], [inherited]

**0.15.183.3.46 `warn()`** OutputWrapper smtrat::parser::InstructionHandler::warn ( ) [inline], [inherited]

#### 0.15.183.4 Field Documentation

**0.15.183.4.1 `infos`** VariantMap<std::string, Value> smtrat::parser::InstructionHandler::infos [protected], [inherited]

**0.15.183.4.2 `lastAnswer`** template<typename Strategy > smtrat::Answer smtrat::Executor< Strategy >::lastAnswer

**0.15.183.4.3 `mDiagnostic`** std::ostream smtrat::parser::InstructionHandler::mDiagnostic [protected], [inherited]

**0.15.183.4.4 `mRegular`** std::ostream smtrat::parser::InstructionHandler::mRegular [protected], [inherited]

**0.15.183.4.5 `options`** VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options [protected], [inherited]

**0.15.183.4.6 `streams`** std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler::streams [protected], [inherited]

### 0.15.184 smtrat::mcsat::fm::Explanation< Settings > Struct Template Reference

#include <Explanation.h>

#### Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &*data*, carl::Variable *var*, const FormulasT &*reason*, bool *force\_use\_core*) const

#### 0.15.184.1 Member Function Documentation

**0.15.184.1.1 `operator()`** template<class Settings > std::optional< mcsat::Explanation > smtrat::mcsat::fm::Explanation< Settings >::operator() ( const mcsat::Bookkeeping & *data*, carl::Variable *var*, const FormulasT & *reason*, bool *force\_use\_core* ) const

### 0.15.185 smtrat::mcsat::icp::Explanation Struct Reference

```
#include <Explanation.h>
```

#### Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &data, carl::Variable var, const FormulasT &reason, bool force\_use\_core) const

#### 0.15.185.1 Member Function Documentation

```
0.15.185.1.1 operator() std::optional< mcsat::Explanation > smtrat::mcsat::icp::Explanation<->::operator() (
 const mcsat::Bookkeeping & data,
 carl::Variable var,
 const FormulasT & reason,
 bool force_use_core) const
```

### 0.15.186 smtrat::mcsat::nlsat::Explanation Struct Reference

```
#include <Explanation.h>
```

#### Public Member Functions

- std::optional< mcsat::Explanation > operator() (const mcsat::Bookkeeping &data, carl::Variable var, const FormulasT &reason, bool) const

We construct a formula ' $E \rightarrow I$ ', i.e.

#### 0.15.186.1 Member Function Documentation

```
0.15.186.1.1 operator() std::optional< mcsat::Explanation > smtrat::mcsat::nlsat::Explanation<->::operator() (
 const mcsat::Bookkeeping & data,
 carl::Variable var,
 const FormulasT & reason,
 bool) const
```

We construct a formula ' $E \rightarrow I$ ', i.e.

'e1 & e2 ... en -> i', called "Explanation", in its clausal form ' $-e_1 \vee -e_2 \dots \vee -e_n \vee i$ ', where e1, e2.. are reason Atoms and CAD cell bound atoms, and i is the implication. It "explains" why the assignment point in data is inconsistent with the reason atoms/constraints and the implication in that it constructs a whole region of assignments points, which the assignment point in data is a part of.

#### Parameters

|                         |                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>variableOrdering</i> | Determine the order of variables, e.g. x1, x2, .. x10                                                                                                                                                                                                                                                                                                    |
| <i>data</i>             | Represent the assigned variables and their assigned values in different representations. These are a prefix of the <i>variableOrdering</i> , e.g. x1, x2, .. x5                                                                                                                                                                                          |
| <i>var</i>              | The smallest/first variable in the <i>variableOrdering</i> that is not assigned by <i>data</i> , e.g. x6                                                                                                                                                                                                                                                 |
| <i>reason</i>           | A collection of atoms that are inconsistent together or imply the <i>implication</i> . These atoms mention only variables x1, x2.. x6 (some atom definitely mentions x6) and possibly x7..x10 if they are irrelevant and "vanish" under the assignment in <i>data</i> , e.g. x1*x7+x6=0 (for x1 := 0) or x6*(x7^2+1)>0 (equiv to x6>0, since x7^2+1 > 0) |

## 0.15.187 `smtrat::mcsat::onecell::Explanation` Struct Reference

```
#include <Explanation.h>
```

### Public Member Functions

- `std::optional< mcsat::Explanation > operator()` (`const mcsat::Bookkeeping &trail, carl::Variable var, const FormulasT &reason, bool`) const

#### 0.15.187.1 Member Function Documentation

```
0.15.187.1.1 operator() std::optional< mcsat::Explanation > smtrat::mcsat::onecell::Explanation::operator() (
 const mcsat::Bookkeeping & trail,
 carl::Variable var,
 const FormulasT & reason,
 bool) const
```

## 0.15.188 `smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1, Setting2 >` Struct Template Reference

```
#include <Explanation.h>
```

### Public Member Functions

- `std::optional< mcsat::Explanation > operator()` (`const mcsat::Bookkeeping &trail, carl::Variable var, const FormulasT &trailLiterals, bool`) const

#### 0.15.188.1 Member Function Documentation

```
0.15.188.1.1 operator() template<class Setting1 , class Setting2 >
std::optional< mcsat::Explanation > smtrat::mcsat::onecellcad::levelwise::Explanation< Setting1,
Setting2 >::operator() (
 const mcsat::Bookkeeping & trail,
 carl::Variable var,
 const FormulasT & trailLiterals,
 bool) const
```

## 0.15.189 `smtrat::mcsat::onecellcad::recursive::Explanation< Setting1, Setting2 >` Struct Template Reference

```
#include <Explanation.h>
```

### Public Member Functions

- `std::optional< mcsat::Explanation > operator()` (`const mcsat::Bookkeeping &trail, carl::Variable var, const FormulasT &trailLiterals, bool`) const

#### 0.15.189.1 Member Function Documentation

```
0.15.189.1.1 operator() template<class Setting1 , class Setting2 >
std::optional< mcsat::Explanation > smtrat::mcsat::onecellcad::recursive::Explanation< Setting1,
Setting2 >::operator() (
 const mcsat::Bookkeeping & trail,
 carl::Variable var,
 const FormulasT & trailLiterals,
 bool) const
```

## 0.15.190 smtrat::mcsat::vs::Explanation Struct Reference

```
#include <Explanation.h>
```

### Public Member Functions

- std::optional< mcsat::Explanation > **operator()** (const mcsat::Bookkeeping &data, carl::Variable var, const FormulasT &reason, bool) const

#### 0.15.190.1 Member Function Documentation

```
0.15.190.1.1 operator() std::optional< mcsat::Explanation > smtrat::mcsat::vs::Explanation< ->::operator() (
 const mcsat::Bookkeeping & data,
 carl::Variable var,
 const FormulasT & reason,
 bool) const
```

## 0.15.191 smtrat::mcsat::nlsat::ExplanationGenerator Class Reference

```
#include <ExplanationGenerator.h>
```

### Public Member Functions

- **ExplanationGenerator** (const std::vector< FormulaT > &constraints, const std::vector< carl::Variable > &vars, carl::Variable, const Model &model)
- void **generateExplanation** (const FormulaT &impliedAtom, std::vector< FormulasT > &explainAtomsInLvl) const
 

*Construct explanation in non-clausal form (without the impliedAtom/implication).*
- mcsat::Explanation **getExplanation** (const FormulaT &impliedAtom) const
 

*Compute explanation in clausal form.*

#### 0.15.191.1 Constructor & Destructor Documentation

```
0.15.191.1.1 ExplanationGenerator() smtrat::mcsat::nlsat::ExplanationGenerator::Explanation< ->::Explanation< ->::Generator (
 const std::vector< FormulaT > & constraints,
 const std::vector< carl::Variable > & vars,
 carl::Variable ,
 const Model & model) [inline]
```

#### 0.15.191.2 Member Function Documentation

```
0.15.191.2.1 generateExplanation() void smtrat::mcsat::nlsat::ExplanationGenerator::generate<Explanation (
```

```
 const FormulaT & impliedAtom,
 std::vector< FormulasT > & explainAtomsInLvl) const [inline]
```

Construct explanation in non-clausal form (without the impliedAtom/implication).

It consists of atomic formulas as expressions for cell component bounds and the constraintAtoms for which a CAD was constructed. Each cell component creates zero, one or two explanation atoms. The impliedAtom is not added to facilitate conversion into a clausal form afterwards.

```
0.15.191.2.2 getExplanation() mcsat::Explanation smtrat::mcsat::nlsat::ExplanationGenerator< ::getExplanation (
```

```
 const FormulaT & impliedAtom) const [inline]
```

Compute explanation in clausal form.

## 0.15.192 smtrat::mcsat::vs::ExplanationGenerator< Settings > Class Template Reference

```
#include <ExplanationGenerator.h>
```

### Public Member Functions

- [ExplanationGenerator](#) (const std::vector< FormulaT > &constraints, const std::vector< carl::Variable > &variableOrdering, const carl::Variable &targetVar, const [Model](#) &model)
- std::optional< [mcsat::Explanation](#) > [getExplanation](#) () const

### 0.15.192.1 Constructor & Destructor Documentation

```
0.15.192.1.1 ExplanationGenerator() template<class Settings >
smtrat::mcsat::vs::ExplanationGenerator< Settings >::ExplanationGenerator (
```

```
 const std::vector< FormulaT > & constraints,
 const std::vector< carl::Variable > & variableOrdering,
 const carl::Variable & targetVar,
 const Model & model) [inline]
```

### 0.15.192.2 Member Function Documentation

```
0.15.192.2.1 getExplanation() template<class Settings >
std::optional< mcsat::Explanation > smtrat::mcsat::vs::ExplanationGenerator< Settings >::get<Explanation () const [inline]
```

## 0.15.193 smtrat::expression::Expression Class Reference

```
#include <Expression.h>
```

### Public Member Functions

- [Expression](#) (carl::Variable::Arg var)
- [Expression](#) (ITEType \_type, [Expression](#) &&\_if, [Expression](#) &&\_then, [Expression](#) &&\_else)
- [Expression](#) (QuantifierType \_type, std::vector< carl::Variable > &&\_variables, [Expression](#) &&\_expression)
- [Expression](#) (UnaryType \_type, [Expression](#) &&\_expression)
- [Expression](#) (UnaryType \_type, const [Expression](#) &\_expression)
- [Expression](#) (BinaryType \_type, [Expression](#) &&\_lhs, [Expression](#) &&\_rhs)
- [Expression](#) (NaryType \_type, [Expressions](#) &&\_expressions)

- Expression (NaryType \_type, const std::initializer\_list< Expression > &\_expressions)
- const ExpressionContent & getContent () const
- const ExpressionContent \* getContentPtr () const
- const ExpressionContent \* getNegationPtr () const
- bool isITE () const
- const ITEExpression & getITE () const
- bool isQuantifier () const
- const QuantifierExpression & getQuantifier () const
- bool isUnary () const
- const UnaryExpression & getUnary () const
- bool isBinary () const
- const BinaryExpression & getBinary () const
- bool is\_nary () const
- const NaryExpression & getNary () const
- bool operator==(const Expression &expr) const
- bool operator< (const Expression &expr) const
- bool operator!= (const Expression &expr) const

**Friends**

- struct std::hash< smtrat::expression::Expression >
- class ExpressionPool
- class ExpressionModifier

**0.15.193.1 Constructor & Destructor Documentation**

**0.15.193.1.1 Expression() [1/8]** smtrat::expression::Expression::Expression ( carl::Variable::Arg var ) [explicit]

**0.15.193.1.2 Expression() [2/8]** smtrat::expression::Expression::Expression ( ITEType \_type,  
Expression && \_if,  
Expression && \_then,  
Expression && \_else ) [explicit]

**0.15.193.1.3 Expression() [3/8]** smtrat::expression::Expression::Expression ( QuantifierType \_type,  
std::vector< carl::Variable > && \_variables,  
Expression && \_expression ) [explicit]

**0.15.193.1.4 Expression() [4/8]** smtrat::expression::Expression::Expression ( UnaryType \_type,  
Expression && \_expression ) [explicit]

**0.15.193.1.5 Expression() [5/8]** smtrat::expression::Expression::Expression ( UnaryType \_type,  
const Expression & \_expression ) [inline], [explicit]

**0.15.193.1.6 Expression()** [6/8] smtrat::expression::Expression ( BinaryType \_type,  
Expression && \_lhs,  
Expression && \_rhs ) [explicit]

**0.15.193.1.7 Expression()** [7/8] smtrat::expression::Expression ( NaryType \_type,  
Expressions && \_expressions ) [explicit]

**0.15.193.1.8 Expression()** [8/8] smtrat::expression::Expression ( NaryType \_type,  
const std::initializer\_list< Expression > & \_expressions ) [explicit]

## 0.15.193.2 Member Function Documentation

**0.15.193.2.1 getBinary()** const BinaryExpression& smtrat::expression::Expression::getBinary ( ) const

**0.15.193.2.2 getContent()** const ExpressionContent& smtrat::expression::Expression::getContent ( ) const [inline]

**0.15.193.2.3 getContentPtr()** const ExpressionContent\* smtrat::expression::Expression::get← ContentPtr ( ) const [inline]

**0.15.193.2.4 getITE()** const ITEExpression& smtrat::expression::Expression::getITE ( ) const

**0.15.193.2.5 getNary()** const NaryExpression& smtrat::expression::Expression::getNary ( ) const

**0.15.193.2.6 getNegationPtr()** const ExpressionContent\* smtrat::expression::Expression::get← NegationPtr ( ) const

**0.15.193.2.7 getQuantifier()** const QuantifierExpression& smtrat::expression::Expression::get← Quantifier ( ) const

**0.15.193.2.8 getUnary()** const UnaryExpression& smtrat::expression::Expression::getUnary ( ) const

**0.15.193.2.9 is\_nary()** bool smtrat::expression::Expression::is\_nary ( ) const

**0.15.193.2.10 isBinary()** bool smtrat::expression::Expression::isBinary ( ) const

**0.15.193.2.11 `isITE()`** `bool smtrat::expression::Expression::isITE ( ) const`

**0.15.193.2.12 `isQuantifier()`** `bool smtrat::expression::Expression::isQuantifier ( ) const`

**0.15.193.2.13 `isUnary()`** `bool smtrat::expression::Expression::isUnary ( ) const`

**0.15.193.2.14 `operator"!="()`** `bool smtrat::expression::Expression::operator!= ( const Expression & expr ) const`

**0.15.193.2.15 `operator<()`** `bool smtrat::expression::Expression::operator< ( const Expression & expr ) const`

**0.15.193.2.16 `operator==()`** `bool smtrat::expression::Expression::operator== ( const Expression & expr ) const`

### 0.15.193.3 Friends And Related Function Documentation

**0.15.193.3.1 `ExpressionModifier`** friend class `ExpressionModifier` [friend]

**0.15.193.3.2 `ExpressionPool`** friend class `ExpressionPool` [friend]

**0.15.193.3.3 `std::hash<smtrat::expression::Expression>`** friend struct `std::hash< smtrat::expression::Expression >` [friend]

## 0.15.194 `smtrat::expression::ExpressionContent` Struct Reference

#include <ExpressionContent.h>

### Public Types

- `typedef boost::variant< carl::Variable, ITEExpression, QuantifierExpression, UnaryExpression, BinaryExpression, NaryExpression > Content`

### Public Member Functions

- `template<typename... T> ExpressionContent (T &&... t)`

### Data Fields

- `Content content`
- `std::size_t id`
- `std::size_t hash`
- `const ExpressionContent * negation`

### Friends

- `struct std::hash< Content >`

### 0.15.194.1 Member Typedef Documentation

**0.15.194.1.1 Content** `typedef boost::variant< carl::Variable, ITEExpression, QuantifierExpression, UnaryExpression, BinaryExpression, NaryExpression > smrat::expression::ExpressionContent::Content`

### 0.15.194.2 Constructor & Destructor Documentation

**0.15.194.2.1 ExpressionContent()** `template<typename... T> smrat::expression::ExpressionContent::ExpressionContent ( T &&... t ) [inline]`

### 0.15.194.3 Friends And Related Function Documentation

**0.15.194.3.1 std::hash< Content >** `friend struct std::hash< Content > [friend]`

### 0.15.194.4 Field Documentation

**0.15.194.4.1 content** `Content smrat::expression::ExpressionContent::content`

**0.15.194.4.2 hash** `std::size_t smrat::expression::ExpressionContent::hash`

**0.15.194.4.3 id** `std::size_t smrat::expression::ExpressionContent::id`

**0.15.194.4.4 negation** `const ExpressionContent* smrat::expression::ExpressionContent::negation`

## 0.15.195 smrat::expression::ExpressionConverter Struct Reference

#include <ExpressionConverter.h>

### Public Member Functions

- `FormulaT operator() (carl::Variable::Arg var)`
- `FormulaT operator() (const ITEExpression &expr)`
- `FormulaT operator() (const QuantifierExpression &expr)`
- `FormulaT operator() (const UnaryExpression &expr)`
- `FormulaT operator() (const BinaryExpression &expr)`
- `FormulaT operator() (const NaryExpression &expr)`
- `FormulaT operator() (const Expression &expr)`

### Protected Member Functions

- `FormulaT convert (const Expression &expr)`

### Protected Attributes

- `FormulasT mGlobalFormulas`

### 0.15.195.1 Member Function Documentation

**0.15.195.1.1 `convert()`** `FormulaT smtrat::expression::ExpressionConverter::convert ( const Expression & expr ) [inline], [protected]`

**0.15.195.1.2 `operator()()` [1/7]** `FormulaT smtrat::expression::ExpressionConverter::operator() ( carl::Variable::Arg var ) [inline]`

**0.15.195.1.3 `operator()()` [2/7]** `FormulaT smtrat::expression::ExpressionConverter::operator() ( const BinaryExpression & expr ) [inline]`

**0.15.195.1.4 `operator()()` [3/7]** `FormulaT smtrat::expression::ExpressionConverter::operator() ( const Expression & expr ) [inline]`

**0.15.195.1.5 `operator()()` [4/7]** `FormulaT smtrat::expression::ExpressionConverter::operator() ( const ITEExpression & expr ) [inline]`

**0.15.195.1.6 `operator()()` [5/7]** `FormulaT smtrat::expression::ExpressionConverter::operator() ( const NaryExpression & expr ) [inline]`

**0.15.195.1.7 `operator()()` [6/7]** `FormulaT smtrat::expression::ExpressionConverter::operator() ( const QuantifierExpression & expr ) [inline]`

**0.15.195.1.8 `operator()()` [7/7]** `FormulaT smtrat::expression::ExpressionConverter::operator() ( const UnaryExpression & expr ) [inline]`

### 0.15.195.2 Field Documentation

**0.15.195.2.1 `mGlobalFormulas`** `FormulasT smtrat::expression::ExpressionConverter::mGlobal<Formulas [protected]`

## 0.15.196 smtrat::expression::ExpressionModifier Class Reference

```
#include <ExpressionVisitor.h>
```

### Public Types

- `typedef std::function< const ExpressionContent *(const Expression &) > VisitorFunction`

### Public Member Functions

- `void setPre (const VisitorFunction &f)`
- `void setPost (const VisitorFunction &f)`
- `Expression visit (const Expression &expression)`
- `const ExpressionContent * operator() (carl::Variable::Arg)`

- const ExpressionContent \* operator() (const ITEExpression &expr)
- const ExpressionContent \* operator() (const QuantifierExpression &expr)
- const ExpressionContent \* operator() (const UnaryExpression &expr)
- const ExpressionContent \* operator() (const BinaryExpression &expr)
- const ExpressionContent \* operator() (const NaryExpression &expr)

### 0.15.196.1 Member Typedef Documentation

**0.15.196.1.1 VisitorFunction** `typedef std::function<const ExpressionContent*(const Expression&)> smrat::expression::ExpressionModifier::VisitorFunction`

### 0.15.196.2 Member Function Documentation

**0.15.196.2.1 operator() [1/6]** `const ExpressionContent* smrat::expression::ExpressionModifier::operator() ( carl::Variable::Arg ) [inline]`

**0.15.196.2.2 operator() [2/6]** `const ExpressionContent* smrat::expression::ExpressionModifier::operator() ( const BinaryExpression & expr ) [inline]`

**0.15.196.2.3 operator() [3/6]** `const ExpressionContent* smrat::expression::ExpressionModifier::operator() ( const ITEExpression & expr ) [inline]`

**0.15.196.2.4 operator() [4/6]** `const ExpressionContent* smrat::expression::ExpressionModifier::operator() ( const NaryExpression & expr ) [inline]`

**0.15.196.2.5 operator() [5/6]** `const ExpressionContent* smrat::expression::ExpressionModifier::operator() ( const QuantifierExpression & expr ) [inline]`

**0.15.196.2.6 operator() [6/6]** `const ExpressionContent* smrat::expression::ExpressionModifier::operator() ( const UnaryExpression & expr ) [inline]`

**0.15.196.2.7 setPost()** `void smrat::expression::ExpressionModifier::setPost ( const VisitorFunction & f ) [inline]`

**0.15.196.2.8 setPre()** `void smrat::expression::ExpressionModifier::setPre ( const VisitorFunction & f ) [inline]`

---

**0.15.196.2.9 visit()** smtrat::expression::ExpressionModifier::visit ( const Expression & expression ) [inline]

## 0.15.197 smtrat::expression::ExpressionPool Class Reference

```
#include <ExpressionPool.h>
```

### Public Member Functions

- const ExpressionContent \* create (carl::Variable::Arg var)
- const ExpressionContent \* create (ITEType \_type, Expression &&\_if, Expression &&\_then, Expression &&\_else)
- const ExpressionContent \* create (QuantifierType \_type, std::vector< carl::Variable > &&\_variables, Expression &&\_expression)
- const ExpressionContent \* create (UnaryType \_type, Expression &&\_expression)
- const ExpressionContent \* create (BinaryType \_type, Expression &&\_lhs, Expression &&\_rhs)
- const ExpressionContent \* create (NaryType \_type, Expressions &&\_expressions)
- const ExpressionContent \* create (NaryType \_type, const std::initializer\_list< Expression > &\_expressions)

### Static Public Member Functions

- template<typename... Args>  
static const ExpressionContent \* create (Args &&... args)

### Protected Member Functions

- ExpressionPool ()
- const ExpressionContent \* add (ExpressionContent \*\_ec)

### Friends

- class ExpressionModifier

## 0.15.197.1 Constructor & Destructor Documentation

**0.15.197.1.1 ExpressionPool()** smtrat::expression::ExpressionPool::ExpressionPool ( ) [inline], [protected]

## 0.15.197.2 Member Function Documentation

**0.15.197.2.1 add()** const ExpressionContent\* smtrat::expression::ExpressionPool::add ( ExpressionContent \* \_ec ) [protected]

**0.15.197.2.2 create() [1/8]** template<typename... Args>  
static const ExpressionContent\* smtrat::expression::ExpressionPool::create ( Args &&... args ) [inline], [static]

**0.15.197.2.3 create() [2/8]** const ExpressionContent\* smtrat::expression::ExpressionPool::create ( BinaryType \_type, Expression && \_lhs, Expression && \_rhs )

```
0.15.197.2.4 create() [3/8] const ExpressionContent* smtrat::expression::ExpressionPool::create
(
 carl::Variable::Arg var)
```

```
0.15.197.2.5 create() [4/8] const ExpressionContent* smtrat::expression::ExpressionPool::create
(
 ITEType _type,
 Expression && _if,
 Expression && _then,
 Expression && _else)
```

```
0.15.197.2.6 create() [5/8] const ExpressionContent* smtrat::expression::ExpressionPool::create
(
 NaryType _type,
 const std::initializer_list< Expression > & _expressions)
```

```
0.15.197.2.7 create() [6/8] const ExpressionContent* smtrat::expression::ExpressionPool::create
(
 NaryType _type,
 Expressions && _expressions)
```

```
0.15.197.2.8 create() [7/8] const ExpressionContent* smtrat::expression::ExpressionPool::create
(
 QuantifierType _type,
 std::vector< carl::Variable > && _variables,
 Expression && _expression)
```

```
0.15.197.2.9 create() [8/8] const ExpressionContent* smtrat::expression::ExpressionPool::create
(
 UnaryType _type,
 Expression && _expression)
```

### 0.15.197.3 Friends And Related Function Documentation

0.15.197.3.1 **ExpressionModifier** friend class ExpressionModifier [friend]

## 0.15.198 smtrat::parser::ParserState::ExpressionScope Struct Reference

```
#include <ParserState.h>
```

### Public Member Functions

- ExpressionScope (const ParserState &state)
- void discharge (ParserState &state)

### 0.15.198.1 Constructor & Destructor Documentation

```
0.15.198.1.1 ExpressionScope() smtrat::parser::ParserState::ExpressionScope::ExpressionScope (
 const ParserState & state) [inline]
```

### 0.15.198.2 Member Function Documentation

```
0.15.198.2.1 discharge() void smtrat::parser::ParserState::ExpressionScope::discharge (
 ParserState & state) [inline]
```

## 0.15.199 smtrat::expression::ExpressionTypeChecker< T > Struct Template Reference

```
#include <ExpressionContent.h>
```

### Public Member Functions

- bool `operator()` (const T &) const
- template<typename T2>  
  bool `operator()` (const T2 &) const

### 0.15.199.1 Member Function Documentation

```
0.15.199.1.1 operator()() [1/2] template<typename T >
bool smtrat::expression::ExpressionTypeChecker< T >::operator() (
 const T &) const [inline]
```

```
0.15.199.1.2 operator()() [2/2] template<typename T >
template<typename T2 >
bool smtrat::expression::ExpressionTypeChecker< T >::operator() (
 const T2 &) const [inline]
```

## 0.15.200 smtrat::expression::ExpressionVisitor Class Reference

```
#include <ExpressionVisitor.h>
```

### Public Types

- `typedef std::function< void(const Expression &) > VisitorFunction`

### Public Member Functions

- void `setPre` (const `VisitorFunction` &f)
- void `setPost` (const `VisitorFunction` &f)
- void `visit` (const `Expression` &expression)
- void `operator()` (carl::Variable::Arg)
- void `operator()` (const `ITEExpression` &expr)
- void `operator()` (const `QuantifierExpression` &expr)
- void `operator()` (const `UnaryExpression` &expr)
- void `operator()` (const `BinaryExpression` &expr)
- void `operator()` (const `NaryExpression` &expr)

### 0.15.200.1 Member Typedef Documentation

**0.15.200.1.1 VisitorFunction** `typedef std::function<void(const Expression&)> smtrat::expression::ExpressionVisitor`

## 0.15.200.2 Member Function Documentation

**0.15.200.2.1 operator() [1/6]** `void smtrat::expression::ExpressionVisitor::operator()( const carl::Variable::Arg & ) [inline]`

**0.15.200.2.2 operator() [2/6]** `void smtrat::expression::ExpressionVisitor::operator()( const BinaryExpression & expr ) [inline]`

**0.15.200.2.3 operator() [3/6]** `void smtrat::expression::ExpressionVisitor::operator()( const ITEExpression & expr ) [inline]`

**0.15.200.2.4 operator() [4/6]** `void smtrat::expression::ExpressionVisitor::operator()( const NaryExpression & expr ) [inline]`

**0.15.200.2.5 operator() [5/6]** `void smtrat::expression::ExpressionVisitor::operator()( const QuantifierExpression & expr ) [inline]`

**0.15.200.2.6 operator() [6/6]** `void smtrat::expression::ExpressionVisitor::operator()( const UnaryExpression & expr ) [inline]`

**0.15.200.2.7 setPost()** `void smtrat::expression::ExpressionVisitor::setPost( const VisitorFunction & f ) [inline]`

**0.15.200.2.8 setPre()** `void smtrat::expression::ExpressionVisitor::setPre( const VisitorFunction & f ) [inline]`

**0.15.200.2.9 visit()** `void smtrat::expression::ExpressionVisitor::visit( const Expression & expression ) [inline]`

## 0.15.201 smtrat::mcsat::FastParallelExplanation< Backends > Struct Template Reference

This explanation executes all given explanation in parallel processes and waits for the fastest explanation, returning the fastest delivered explanation, terminating all other parallel processes.

```
#include <FastParallelExplanation.h>
```

### 0.15.201.1 Detailed Description

```
template<typename... Backends>
struct smtrat::mcsat::FastParallelExplanation< Backends >
```

This explanation executes all given explanation in parallel processes and waits for the fastest explanation, returning the fastest delivered explanation, terminating all other parallel processes.

## 0.15.202 smtrat::mcsat::variableordering::detail::FeatureCollector< Objects > Struct Template Reference

This class manages features that are used to valuate variables on objects.

```
#include <feature_based.h>
```

### Public Types

- using `Extractor` = std::function< double(Objects, carl::Variable) >
- using `Valuation` = std::vector< double >

### Public Member Functions

- void `addFeature (Extractor &&e, std::size_t level, double weight)`
- `Valuation valuateVariable (const Objects &o, carl::Variable v) const`
- `std::vector< carl::Variable > sortVariables (const Objects &o, std::vector< carl::Variable > vars) const`

### Data Fields

- `std::vector< std::tuple< Extractor, std::size_t, double > > mFeatures`

#### 0.15.202.1 Detailed Description

```
template<typename Objects>
struct smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >
```

This class manages features that are used to valuate variables on objects.

Each feature consists of a valuation function, a level and a weight. All feature valuations of a certain level are combined linearly using the respective weights. Valuations are then compared lexicographically.

#### 0.15.202.2 Member Typedef Documentation

```
0.15.202.2.1 Extractor template<typename Objects >
using smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::Extractor = std::function<double(Objects, carl::Variable)>
```

```
0.15.202.2.2 Valuation template<typename Objects >
using smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::Valuation = std::vector<double>
```

#### 0.15.202.3 Member Function Documentation

```
0.15.202.3.1 addFeature() template<typename Objects >
void smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::addFeature (
 Extractor && e,
 std::size_t level,
 double weight) [inline]
```

```
0.15.202.3.2 sortVariables() template<typename Objects >
std::vector<carl::Variable> smtrat::mcsat::variableordering::detail::FeatureCollector< Objects
>::sortVariables (
 const Objects & o,
 std::vector< carl::Variable > vars) const [inline]
```

```
0.15.202.3.3 evaluateVariable() template<typename Objects >
Valuation smtrat::mcsat::variableordering::detail::FeatureCollector< Objects >::evaluate<-
Variable (
 const Objects & o,
 carl::Variable v) const [inline]
```

## 0.15.202.4 Field Documentation

```
0.15.202.4.1 mFeatures template<typename Objects >
std::vector<std::tuple<Extractor, std::size_t, double> > smtrat::mcsat::variableordering::detail::FeatureColle-
Objects >::mFeatures
```

## 0.15.203 smtrat::fixedsize\_allocator< T > Class Template Reference

An allocator meant to be used for containers that typically allocate single objects or nodes, such as sets, maps and linked lists.

```
#include <alloc.h>
```

### Data Structures

- struct [rebind](#)

### Public Member Functions

- [fixedsize\\_allocator \(\) NOEXCEPT](#)
- [fixedsize\\_allocator \(const fixedsize\\_allocator &\) NOEXCEPT=default](#)
- [template<typename U > fixedsize\\_allocator \(const fixedsize\\_allocator< U > &\) NOEXCEPT](#)
- [template<typename T2 > CONSTEXPR bool operator==\(const fixedsize\\_allocator< T2 > &\) const NOEXCEPT](#)
- [template<typename T2 > CONSTEXPR bool operator!=\(const fixedsize\\_allocator< T2 > &\) const NOEXCEPT](#)

### 0.15.203.1 Detailed Description

```
template<typename T>
class smtrat::fixedsize_allocator< T >
```

An allocator meant to be used for containers that typically allocate single objects or nodes, such as sets, maps and linked lists.

For any allocation that allocates more than one element at once, this falls back to std::allocator. With this allocator, free consists of two single pointer operations only. Allocate is almost always as cheap as free.

### 0.15.203.2 Constructor & Destructor Documentation

```
0.15.203.2.1 fixedsize_allocator() [1/3] template<typename T >
smtrat::fixedsize_allocator< T >::fixedsize_allocator () [inline]
```

```
0.15.203.2.2 fixedsize_allocator() [2/3] template<typename T >
smtrat::fixedsize_allocator< T >::fixedsize_allocator (
 const fixedsize_allocator< T > &) [default]
```

```
0.15.203.2.3 fixedsize_allocator() [3/3] template<typename T >
template<typename U >
smtrat::fixedsize_allocator< T >::fixedsize_allocator (
 const fixedsize_allocator< U > &) [inline]
```

### 0.15.203.3 Member Function Documentation

```
0.15.203.3.1 operator"!=() template<typename T >
template<typename T2 >
CONSTEXPR bool smtrat::fixedsize_allocator< T >::operator!= (
 const fixedsize_allocator< T2 > &) const [inline]
```

```
0.15.203.3.2 operator==() template<typename T >
template<typename T2 >
CONSTEXPR bool smtrat::fixedsize_allocator< T >::operator== (
 const fixedsize_allocator< T2 > &) const [inline]
```

## 0.15.204 smtrat::fixedsize\_allocator< void > Class Reference

I do not know where an allocator for void would be of any use; but in order to mimic std::allocator (where this is explicitly mentioned) as good as possible, we do this for the void case.

```
#include <alloc.h>
```

### Data Fields

- **T elements**

*STL member.*

### 0.15.204.1 Detailed Description

I do not know where an allocator for void would be of any use; but in order to mimic std::allocator (where this is explicitly mentioned) as good as possible, we do this for the void case.

### 0.15.204.2 Field Documentation

```
0.15.204.2.1 elements T std::allocator< T >::elements [inherited]
STL member.
```

## 0.15.205 smtrat::impl::fixedsize\_allocator\_freelists Class Reference

```
#include <alloc.h>
```

### Static Public Attributes

- static **fixedsize\_allocator\_freelists INSTANCE**

### 0.15.205.1 Field Documentation

**0.15.205.1.1 INSTANCE `fixedsize_allocator_freelists` `smtrat::impl::fixedsize_allocator_` ↵  
freelists::INSTANCE [static]**

**0.15.206 `smtrat::impl::fixedsize_allocator_impl< T, SizeShift, LargeEnough,  
SmallEnough >` Class Template Reference**

**0.15.207 `smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >` Class  
Template Reference**

```
#include <alloc.h>
```

#### Public Types

- `typedef T value_type`
- `typedef T * pointer`
- `typedef const T * const_pointer`
- `typedef T & reference`
- `typedef const T & const_reference`
- `typedef std::size_t size_type`
- `typedef std::ptrdiff_t difference_type`
- `typedef std::true_type propagate_on_container_move_assignment`
- `typedef std::true_type is_always_equal`

#### Public Member Functions

- `pointer address (reference x) const NOEXCEPT`
- `const_reference address (const_reference x) const NOEXCEPT`
- `CONSTEXPR size_type max_size () const NOEXCEPT`
- `pointer allocate (size_type n, const void *hint=0)`
- `void deallocate (pointer p, size_type n)`
- `template<class U , class... Args>  
void construct (U *p, Args &&... args)`
- `template<class U >  
void destroy (U *p)`

### 0.15.207.1 Member Typedef Documentation

**0.15.207.1.1 `const_pointer` template<typename T , std::size\_t SizeShift>  
`typedef const T* smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::const_pointer`**

**0.15.207.1.2 `const_reference` template<typename T , std::size\_t SizeShift>  
`typedef const T& smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::const_reference`**

**0.15.207.1.3 `difference_type` template<typename T , std::size\_t SizeShift>  
`typedef std::ptrdiff_t smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::difference_type`**

**0.15.207.1.4 `is_always_equal`** template<typename T , std::size\_t SizeShift>  
typedef std::true\_type smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::  
::is\_always\_equal

**0.15.207.1.5 `pointer`** template<typename T , std::size\_t SizeShift>  
typedef T\* smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::pointer

**0.15.207.1.6 `propagate_on_container_move_assignment`** template<typename T , std::size\_t SizeShift>  
typedef std::true\_type smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::  
::propagate\_on\_container\_move\_assignment

**0.15.207.1.7 `reference`** template<typename T , std::size\_t SizeShift>  
typedef T& smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::reference

**0.15.207.1.8 `size_type`** template<typename T , std::size\_t SizeShift>  
typedef std::size\_t smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::size\_type

**0.15.207.1.9 `value_type`** template<typename T , std::size\_t SizeShift>  
typedef T smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::value\_type

## 0.15.207.2 Member Function Documentation

**0.15.207.2.1 `address()` [1/2]** template<typename T , std::size\_t SizeShift>  
const\_reference smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::address ( const\_reference x ) const [inline]

**0.15.207.2.2 `address()` [2/2]** template<typename T , std::size\_t SizeShift>  
pointer smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::address ( reference x ) const [inline]

**0.15.207.2.3 `allocate()`** template<typename T , std::size\_t SizeShift>  
pointer smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::allocate ( size\_type n, const void \* hint = 0 ) [inline]

**0.15.207.2.4 `construct()`** template<typename T , std::size\_t SizeShift>  
template<class U , class... Args>  
void smtrat::impl::fixedsize\_allocator\_impl< T, SizeShift, true, true >::construct ( U \* p, Args &&... args ) [inline]

```
0.15.207.2.5 deallocate() template<typename T , std::size_t SizeShift>
void smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::deallocate (
 pointer p,
 size_type n) [inline]
```

```
0.15.207.2.6 destroy() template<typename T , std::size_t SizeShift>
template<class U >
void smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::destroy (
 U * p) [inline]
```

```
0.15.207.2.7 max_size() template<typename T , std::size_t SizeShift>
CONSTEXPR size_type smtrat::impl::fixedsize_allocator_impl< T, SizeShift, true, true >::max_size () const [inline]
```

## 0.15.208 smtrat::impl::fixedsize\_allocator\_size\_helper< T, Size > Class Template Reference

```
#include <alloc.h>
```

## 0.15.209 smtrat::fixedsize\_freelist< ChunkSizeShift > Class Template Reference

Implements a fixed-(at-compile-time)-size allocator based on a free list embedded in the space of currently unused objects.

```
#include <alloc.h>
```

### Public Member Functions

- **fixedsize\_freelist ()**
- **~fixedsize\_freelist ()**
- **void \* alloc ()**  
*Allocate an object of size 1<<ChunkSizeShift.*
- **void free (void \*ptr) NOEXCEPT**  
*Freeing means just prepending the free'd element to the front of the freelist.*

### 0.15.209.1 Detailed Description

```
template<std::size_t ChunkSizeShift>
class smtrat::fixedsize_freelist< ChunkSizeShift >
```

Implements a fixed-(at-compile-time)-size allocator based on a free list embedded in the space of currently unused objects.

### 0.15.209.2 Constructor & Destructor Documentation

```
0.15.209.2.1 fixedsize_freelist() template<std::size_t ChunkSizeShift>
smtrat::fixedsize_freelist< ChunkSizeShift >::fixedsize_freelist () [inline], [explicit]
```

```
0.15.209.2.2 ~fixedsize_freelist() template<std::size_t ChunkSizeShift>
smtrat::fixedsize_freelist< ChunkSizeShift >::~fixedsize_freelist () [inline]
```

### 0.15.209.3 Member Function Documentation

**0.15.209.3.1 alloc()** template<std::size\_t ChunkSizeShift>  
void\* smtrat::fixedsize\_freelist< ChunkSizeShift >::alloc ( ) [inline]  
Allocate an object of size  $1 \ll \text{ChunkSizeShift}$ .  
First, try to return the first entry on the freelist.  
If that is empty, try forward-allocation (allocating the next element on the current page that has never been allocated before.)  
Otherwise, go to the next page.  
If there is no next page, allocate a new page with malloc. This page has twice the space of the previous page.  
Start forward-allocation phase on the new page.

**0.15.209.3.2 free()** template<std::size\_t ChunkSizeShift>  
void smtrat::fixedsize\_freelist< ChunkSizeShift >::free ( void \* ptr ) [inline]

Freeling means just prepending the free'd element to the front of the freelist.

## 0.15.210 smtrat::impl::fixedsize\_freelists< SizeCurrent, SizeMax, LowEnough > Class Template Reference

Publicly derive from all [fixedsize\\_freelist](#) types with chunk sizes between SizeCurrent and SizeMax.

```
#include <alloc.h>
```

### 0.15.210.1 Detailed Description

```
template<std::size_t SizeCurrent, std::size_t SizeMax, bool LowEnough = (SizeCurrent < SizeMax)>
class smtrat::impl::fixedsize_freelists< SizeCurrent, SizeMax, LowEnough >
```

Publicly derive from all [fixedsize\\_freelist](#) types with chunk sizes between SizeCurrent and SizeMax.

## 0.15.211 smtrat::impl::fixedsize\_freelists< SizeCurrent, SizeMax, true > Class Template Reference

```
#include <alloc.h>
```

## 0.15.212 smtrat::parser::FixedWidthConstant< T > Struct Template Reference

Represents a constant of a fixed width.

```
#include <TheoryTypes.h>
```

### Public Member Functions

- [FixedWidthConstant \(\)](#)
- [FixedWidthConstant \(const T &value, std::size\\_t width\)](#)
- [bool operator< \(const FixedWidthConstant &fwc\) const](#)

### Data Fields

- [T value](#)
- [std::size\\_t width](#)

### 0.15.212.1 Detailed Description

```
template<typename T>
struct smtrat::parser::FixedWidthConstant< T >
```

Represents a constant of a fixed width.

### 0.15.212.2 Constructor & Destructor Documentation

**0.15.212.2.1 FixedWidthConstant()** [1/2] template<typename T >  
smtrat::parser::FixedWidthConstant< T >::FixedWidthConstant ( ) [inline]

**0.15.212.2.2 FixedWidthConstant()** [2/2] template<typename T >  
smtrat::parser::FixedWidthConstant< T >::FixedWidthConstant (  
    const T & value,  
    std::size\_t width ) [inline]

### 0.15.212.3 Member Function Documentation

**0.15.212.3.1 operator<()** template<typename T >  
bool smtrat::parser::FixedWidthConstant< T >::operator< (  
    const FixedWidthConstant< T > & fwc ) const [inline]

### 0.15.212.4 Field Documentation

**0.15.212.4.1 value** template<typename T >  
T smtrat::parser::FixedWidthConstant< T >::value

**0.15.212.4.2 width** template<typename T >  
std::size\_t smtrat::parser::FixedWidthConstant< T >::width

## 0.15.213 smtrat::parseformula::FormulaCollector Class Reference

```
#include <parser_smtlib_utils.h>
```

### Public Types

- typedef types::AttributeValue Value

### Public Member Functions

- void add (const smtrat::FormulaT &f)
- void addSoft (const FormulaT &f, Rational, const std::string &)
- void annotateName (const smtrat::FormulaT &, const std::string &)
- void check ()
- void declareFun (const carl::Variable &)
- void declareSort (const std::string &, const unsigned &)
- void defineSort (const std::string &, const std::vector< std::string > &, const carl::Sort &)
- void eliminateQuantifiers (const smtrat::qe::QEQuery &)
- void exit ()
- void getAssertions ()
- void getAllModels ()
- void getAssignment ()
- void getModel ()
- void getObjectives ()
- void getProof ()
- void getUnsatCore ()
- void getValue (const std::vector< carl::Variable > &)
- void addObjective (const smtrat::Poly &, smtrat::parser::OptimizationType)
- void pop (std::size\_t)

- void `push` (std::size\_t)
- void `reset` ()
- void `resetAssertions` ()
- void `setLogic` (const carl::Logic &l)
- int `getExitCode` () const
- `FormulaT getFormula` () const
- auto `getLogic` () const
- void `addInstruction` (std::function< void()> bind)
- bool `hasInstructions` () const
- void `runInstructions` ()
- void `setArtificialVariables` (std::vector< smrat::ModelVariable > &&vars)
- void `cleanModel` (smrat::Model &model) const
- bool `has_info` (const std::string &key) const
- const auto & `get_info` (const std::string &key) const
- template<typename T >  
    T `option` (const std::string &key) const
- bool `printInstruction` () const
- std::ostream & `diagnostic` ()
- void `diagnostic` (const std::string &s)
- std::ostream & `regular` ()
- void `regular` (const std::string &s)
- OutputWrapper `error` ()
- OutputWrapper `warn` ()
- OutputWrapper `info` ()
- virtual void `echo` (const std::string &s)
- void `getInfo` (const std::string &key)
- void `getOption` (const std::string &key)
- void `setInfo` (const Attribute &attr)
- void `setOption` (const Attribute &option)

### Protected Member Functions

- void `setStream` (const std::string &s, std::ostream &os)

### Protected Attributes

- VariantMap< std::string, Value > infos
- VariantMap< std::string, Value > options
- std::ostream `mRegular`
- std::ostream `mDiagnostic`
- std::map< std::string, std::ofstream > `streams`

### 0.15.213.1 Member Typedef Documentation

**0.15.213.1.1 Value** `typedef types::AttributeValue smrat::parser::InstructionHandler::Value`  
 [inherited]

### 0.15.213.2 Member Function Documentation

**0.15.213.2.1 add()** `void smrat::parseformula::FormulaCollector::add (`  
`const smrat::FormulaT & f ) [inline], [virtual]`  
 Implements `smrat::parser::InstructionHandler`.

**0.15.213.2.2 addInstruction()** void smtrat::parser::InstructionHandler::addInstruction ( std::function< void()> bind ) [inline], [inherited]

**0.15.213.2.3 addObjective()** void smtrat::parseformula::FormulaCollector::addObjective ( const smtrat::Poly &, smtrat::parser::OptimizationType ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.4 addSoft()** void smtrat::parseformula::FormulaCollector::addSoft ( const FormulaT & f, Rational, const std::string & ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.5 annotateName()** void smtrat::parseformula::FormulaCollector::annotateName ( const smtrat::FormulaT &, const std::string & ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.6 check()** void smtrat::parseformula::FormulaCollector::check ( ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.7 cleanModel()** void smtrat::parser::InstructionHandler::cleanModel ( smtrat::Model & model ) const [inline], [inherited]

**0.15.213.2.8 declareFun()** void smtrat::parseformula::FormulaCollector::declareFun ( const carl::Variable & ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.9 declareSort()** void smtrat::parseformula::FormulaCollector::declareSort ( const std::string &, const unsigned & ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.10 defineSort()** void smtrat::parseformula::FormulaCollector::defineSort ( const std::string &, const std::vector< std::string > &, const carl::Sort & ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.11 diagnostic() [1/2]** std::ostream& smtrat::parser::InstructionHandler::diagnostic ( ) [inline], [inherited]

**0.15.213.2.12 diagnostic() [2/2]** void smtrat::parser::InstructionHandler::diagnostic ( const std::string & s ) [inline], [inherited]

**0.15.213.2.13 echo()** virtual void smtrat::parser::InstructionHandler::echo ( const std::string & s ) [inline], [virtual], [inherited]

**0.15.213.2.14 eliminateQuantifiers()** void smtrat::parseformula::FormulaCollector::eliminate← Quantifiers ( const smtrat::qe::QEQuery & ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.15 error()** OutputWrapper smtrat::parser::InstructionHandler::error ( ) [inline], [inherited]

**0.15.213.2.16 exit()** void smtrat::parseformula::FormulaCollector::exit ( ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.17 get\_info()** const auto& smtrat::parser::InstructionHandler::get\_info ( const std::string & key ) const [inline], [inherited]

**0.15.213.2.18 getAllModels()** void smtrat::parseformula::FormulaCollector::getAllModels ( ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.19 getAssertions()** void smtrat::parseformula::FormulaCollector::getAssertions ( ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.20 getAssignment()** void smtrat::parseformula::FormulaCollector::getAssignment ( ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.21 getExitCode()** int smtrat::parseformula::FormulaCollector::getExitCode ( ) const [inline]

**0.15.213.2.22 getFormula()** [FormulaT](#) smtrat::parseformula::FormulaCollector::getFormula ( ) const [inline]

**0.15.213.2.23 getInfo()** void smtrat::parser::InstructionHandler::getInfo ( const std::string & key ) [inline], [inherited]

**0.15.213.2.24 getLogic()** auto smtrat::parseformula::FormulaCollector::getLogic ( ) const [inline]

**0.15.213.2.25 getModel()** void smtrat::parseformula::FormulaCollector::getModel ( ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.26 `getObjectives()`** void smrat::parseformula::FormulaCollector::getObjectives ( )  
[inline], [virtual]  
Implements [smrat::parser::InstructionHandler](#).

**0.15.213.2.27 `getOption()`** void smrat::parser::InstructionHandler::getOption ( const std::string & key ) [inline], [inherited]

**0.15.213.2.28 `getProof()`** void smrat::parseformula::FormulaCollector::getProof ( ) [inline], [virtual]  
Implements [smrat::parser::InstructionHandler](#).

**0.15.213.2.29 `getUnsatCore()`** void smrat::parseformula::FormulaCollector::getUnsatCore ( )  
[inline], [virtual]  
Implements [smrat::parser::InstructionHandler](#).

**0.15.213.2.30 `getValue()`** void smrat::parseformula::FormulaCollector::getValue ( const std::vector< carl::Variable > & ) [inline], [virtual]  
Implements [smrat::parser::InstructionHandler](#).

**0.15.213.2.31 `has_info()`** bool smrat::parser::InstructionHandler::has\_info ( const std::string & key ) const [inline], [inherited]

**0.15.213.2.32 `hasInstructions()`** bool smrat::parser::InstructionHandler::hasInstructions ( ) const [inline], [inherited]

**0.15.213.2.33 `info()`** OutputWrapper smrat::parser::InstructionHandler::info ( ) [inline], [inherited]

**0.15.213.2.34 `option()`** template<typename T >  
T smrat::parser::InstructionHandler::option ( const std::string & key ) const [inline], [inherited]

**0.15.213.2.35 `pop()`** void smrat::parseformula::FormulaCollector::pop ( std::size\_t ) [inline], [virtual]  
Implements [smrat::parser::InstructionHandler](#).

**0.15.213.2.36 `printInstruction()`** bool smrat::parser::InstructionHandler::printInstruction ( ) const [inline], [inherited]

**0.15.213.2.37 `push()`** void smrat::parseformula::FormulaCollector::push ( std::size\_t ) [inline], [virtual]  
Implements [smrat::parser::InstructionHandler](#).

**0.15.213.2.38 regular() [1/2]** std::ostream& smtrat::parser::InstructionHandler::regular ( )  
[inline], [inherited]

**0.15.213.2.39 regular() [2/2]** void smtrat::parser::InstructionHandler::regular ( const std::string & s ) [inline], [inherited]

**0.15.213.2.40 reset()** void smtrat::parseformula::FormulaCollector::reset ( ) [inline], [virtual]  
Reimplemented from [smtrat::parser::InstructionHandler](#).

**0.15.213.2.41 resetAssertions()** void smtrat::parseformula::FormulaCollector::resetAssertions ( )  
[inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.42 runInstructions()** void smtrat::parser::InstructionHandler::runInstructions ( )  
[inline], [inherited]

**0.15.213.2.43 setArtificialVariables()** void smtrat::parser::InstructionHandler::setArtificialVariables ( std::vector< [smtrat::ModelVariable](#) > && vars ) [inline], [inherited]

**0.15.213.2.44 setInfo()** void smtrat::parser::InstructionHandler::setInfo ( const [Attribute](#) & attr ) [inline], [inherited]

**0.15.213.2.45 setLogic()** void smtrat::parseformula::FormulaCollector::setLogic ( const carl::Logic & l ) [inline], [virtual]  
Implements [smtrat::parser::InstructionHandler](#).

**0.15.213.2.46 setOption()** void smtrat::parser::InstructionHandler::setOption ( const [Attribute](#) & option ) [inline], [inherited]

**0.15.213.2.47 setStream()** void smtrat::parser::InstructionHandler::setStream ( const std::string & s, std::ostream & os ) [inline], [protected], [inherited]

**0.15.213.2.48 warn()** OutputWrapper smtrat::parser::InstructionHandler::warn ( ) [inline], [inherited]

### 0.15.213.3 Field Documentation

**0.15.213.3.1 infos** [VariantMap](#)<std::string, [Value](#)> smtrat::parser::InstructionHandler::infos  
[protected], [inherited]

**0.15.213.3.2 mDiagnostic** std::ostream smtrat::parser::InstructionHandler::mDiagnostic [protected], [inherited]

**0.15.213.3.3 mRegular** std::ostream smtrat::parser::InstructionHandler::mRegular [protected], [inherited]

**0.15.213.3.4 options** VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options [protected], [inherited]

**0.15.213.3.5 streams** std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler::streams [protected], [inherited]

## 0.15.214 smtrat::ICPModule< Settings >::formulaPtrComp Struct Reference

```
#include <ICPModule.h>
```

### Public Member Functions

- bool [operator\(\)](#) (const [FormulaT](#) &\_lhs, const [FormulaT](#) &\_rhs) const

### 0.15.214.1 Member Function Documentation

**0.15.214.1.1 operator()** template<class Settings >  
bool smtrat::ICPModule< Settings >::formulaPtrComp::operator() (  
 const [FormulaT](#) & \_lhs,  
 const [FormulaT](#) & \_rhs ) const [inline]

## 0.15.215 smtrat::FormulaWithOrigins Class Reference

Stores a formula along with its origins.

```
#include <ModuleInput.h>
```

### Public Member Functions

- [FormulaWithOrigins \(\)](#)
- [FormulaWithOrigins \(const FormulaT &\\_formula\)](#)  
*Constructs a formula with empty origins.*
- [FormulaWithOrigins \(const FormulaT &\\_formula, const std::shared\\_ptr< FormulasT > &\\_origins\)](#)  
*Constructs a formula with the given origins.*
- [FormulaWithOrigins \(const FormulaWithOrigins &\)](#)
- const [FormulaT & formula \(\) const](#)
- bool [hasOrigins \(\) const](#)
- const [FormulasT & origins \(\) const](#)
- void [setDeducted \(bool \\_deducted\) const](#)  
*Sets the deduction flag to the given value.*
- bool [deducted \(\) const](#)

### Friends

- class [ModuleInput](#)
- bool [operator< \(const FormulaWithOrigins &\\_fwoA, const FormulaWithOrigins &\\_fwoB\)](#)
- bool [operator== \(const FormulaWithOrigins &\\_fwoA, const FormulaWithOrigins &\\_fwoB\)](#)

### 0.15.215.1 Detailed Description

Stores a formula along with its origins.

### 0.15.215.2 Constructor & Destructor Documentation

#### 0.15.215.2.1 `FormulaWithOrigins()` [1/4] `smtrat::FormulaWithOrigins::FormulaWithOrigins()`

```
0.15.215.2.2 FormulaWithOrigins() [2/4] smtrat::FormulaWithOrigins::FormulaWithOrigins(
 const FormulaT & _formula) [inline]
```

Constructs a formula with empty origins.

#### Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <code>_formula</code> | The formula of the formula with origins to construct. |
|-----------------------|-------------------------------------------------------|

#### 0.15.215.2.3 `FormulaWithOrigins()` [3/4] `smtrat::FormulaWithOrigins::FormulaWithOrigins(`

```
 const FormulaT & _formula,
 const std::shared_ptr< FormulasT > & _origins) [inline]
```

Constructs a formula with the given origins.

#### Parameters

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| <code>_formula</code> | The formula of the formula with origins to construct. |
| <code>_origins</code> | The origins of the formula with origins to construct. |

#### 0.15.215.2.4 `FormulaWithOrigins()` [4/4] `smtrat::FormulaWithOrigins::FormulaWithOrigins(`

```
 const FormulaWithOrigins &)
```

### 0.15.215.3 Member Function Documentation

#### 0.15.215.3.1 `deducted()` `bool smtrat::FormulaWithOrigins::deducted()` const [inline]

#### Returns

The deduction flag, which indicates, that this formula g is a direct sub-formula of a conjunction of formulas (and g f\_1 .. f\_n), and, that (implies (and f\_1 .. f\_n) g) holds.

#### 0.15.215.3.2 `formula()` `const FormulaT& smtrat::FormulaWithOrigins::formula()` const [inline]

#### Returns

A constant reference to the formula.

**0.15.215.3.3 hasOrigins()** `bool smtrat::FormulaWithOrigins::hasOrigins () const [inline]`**Returns**

true, if this sub-formula of the module input has origins.

**0.15.215.3.4 origins()** `const FormulasT& smtrat::FormulaWithOrigins::origins () const [inline]`**Returns**

A constant reference to the origins.

**0.15.215.3.5 setDeducted()** `void smtrat::FormulaWithOrigins::setDeducted (bool _deducted) const [inline]`

Sets the deduction flag to the given value.

**Parameters**

|                        |                                         |
|------------------------|-----------------------------------------|
| <code>_deducted</code> | The value to set the deduction flag to. |
|------------------------|-----------------------------------------|

**0.15.215.4 Friends And Related Function Documentation****0.15.215.4.1 ModuleInput** `friend class ModuleInput [friend]`**0.15.215.4.2 operator<** `bool operator< (const FormulaWithOrigins & _fwoA, const FormulaWithOrigins & _fwoB) [friend]`**Parameters**

|                    |                                             |
|--------------------|---------------------------------------------|
| <code>_fwoA</code> | The first formula with origins to compare.  |
| <code>_fwoB</code> | The second formula with origins to compare. |

**Returns**

true, if the formula of the first formula with origins is less than the formula of the second formula with origins; false, otherwise.

**0.15.215.4.3 operator==** `bool operator== (const FormulaWithOrigins & _fwoA, const FormulaWithOrigins & _fwoB) [friend]`**Parameters**

|                    |                                             |
|--------------------|---------------------------------------------|
| <code>_fwoA</code> | The first formula with origins to compare.  |
| <code>_fwoB</code> | The second formula with origins to compare. |

**Returns**

true, if the formula of the first formula with origins and the formula of the second formula with origins are equal;  
false, otherwise.

## 0.15.216 smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices > Class Template Reference

This class implements a forward hypergraph.

```
#include <ForwardHyperGraph.h>
```

### Data Structures

- struct [Edge](#)

*Internal type of an edge.*

### Public Types

- using [Vertex](#) = std::size\_t

*Internal type of a vertex.*

### Public Member Functions

- [Vertex newVertex \(const VertexProperty &vp\)](#)

*Returns a new vertex that is labeled with the given vertex property.*

- [Edge & newEdge \(Vertex source, const EdgeProperty &ep\)](#)

*Creates a new edge where the given vertex is the incoming vertex that is labeled with the given edge property.*

- [Edge & newEdge \(const VertexProperty &source, const EdgeProperty &ep\)](#)

*Creates a new edge where the given vertex is the incoming vertex that is labeled with the given edge property.*

- const VertexProperty & [operator\[\] \(Vertex v\) const](#)

*Retrieves the vertex property for the given vertex.*

- VertexProperty & [operator\[\] \(Vertex v\)](#)

*Retrieves the vertex property for the given vertex.*

- const std::vector< [Edge](#) > & [out \(Vertex v\) const](#)

*Retrieves all outgoing edges for the given vertex.*

- const std::vector< [Edge](#) > & [out \(const VertexProperty &v\) const](#)

*Retrieves all outgoing edges for the given vertex.*

- [Vertex begin \(\) const](#)

*Returns the first vertex.*

- [Vertex end \(\) const](#)

*Returns the past-the-last vertex.*

- std::size\_t [size \(\) const](#)

*Returns the number of vertices.*

- void [toDot \(const std::string &filename\) const](#)

*Writes a representation of this graph to the given file.*

### Friends

- template<typename VP , typename EP >  
std::ostream & [operator<< \(std::ostream &os, const ForwardHyperGraph< VP, EP > &fhg\)](#)

*Writes a textual representation of this graph to the given stream.*

### 0.15.216.1 Detailed Description

```
template<typename VertexProperty, typename EdgeProperty, bool UniqueVertices = true>
class smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >
```

This class implements a forward hypergraph.

We define a hypergraph to be a graph structure \$(V,E)\$ where \$V\$ is a set of vertices and \$E \subseteq 2^V\$ a set of hyperedges. A hyperedge can connect an arbitrary number of vertices.

If a hypergraph is directed if every edge \$e \in E\$ is partitioned into \$I(e)\$ and \$O(e)\$ that are sets of incoming vertices and outgoing vertices, respectively. A forward hypergraph is directed and every edge has only a single incoming vertices, that is \$|I(e)| = 1\$. Note that we store edges as a mapping from the incoming vertex to a set of outgoing vertices and thus do not have an explicit incoming vertex in our edge representation.

This implementation of a forward hypergraph accepts arbitrary labels (called properties) for both vertices and edges. It relies on a dense list of vertices that are internally labeled starting from zero. Therefore, removal of vertices is not supported.

If the third template parameter UniqueVertices is set to true (being the default), we make sure that every vertex is unique with respect to its property. To make this possible, a vertex property may not be changed in this case which is enforced using a static assertion. No requirements are imposed on the edge properties.

This class supports both printing a simple text representation and writing a description to a dot file.

### 0.15.216.2 Member Typedef Documentation

**0.15.216.2.1 Vertex** `template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>`  
`using smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::Vertex =`  
`std::size_t`  
Internal type of a vertex.

### 0.15.216.3 Member Function Documentation

**0.15.216.3.1 begin()** `template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>`  
`Vertex smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::begin ()`  
const [inline]  
Returns the first vertex.

**0.15.216.3.2 end()** `template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>`  
`Vertex smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::end ()`  
const [inline]  
Returns the past-the-last vertex.

**0.15.216.3.3 newEdge() [1/2]** `template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>`  
`Edge& smrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::newEdge (`  
const VertexProperty & source,  
const EdgeProperty & ep ) [inline]  
Creates a new edge where the given vertex is the incoming vertex that is labeled with the given edge property.

```
0.15.216.3.4 newEdge() [2/2] template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
Edge& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::newEdge (
 Vertex source,
 const EdgeProperty & ep) [inline]
```

Creates a new edge where the given vertex is the incoming vertex that is labeled with the given edge property.

```
0.15.216.3.5 newVertex() template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
Vertex smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::newVertex (
 const VertexProperty & vp) [inline]
```

Returns a new vertex that is labeled with the given vertex property.

If UniqueVertices is set to true and a vector with the given property already exists, this vector is returned.

```
0.15.216.3.6 operator[]() [1/2] template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
VertexProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::operator[] (
 Vertex v) [inline]
```

Retrieves the vertex property for the given vertex.

This method is disabled if UniqueVertices is set to true.

```
0.15.216.3.7 operator[]() [2/2] template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
const VertexProperty& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::operator[] (
 Vertex v) const [inline]
```

Retrieves the vertex property for the given vertex.

```
0.15.216.3.8 out() [1/2] template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
const std::vector<Edge>& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::out (
 const VertexProperty & v) const [inline]
```

Retrieves all outgoing edges for the given vertex.

```
0.15.216.3.9 out() [2/2] template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
const std::vector<Edge>& smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::out (
 Vertex v) const [inline]
```

Retrieves all outgoing edges for the given vertex.

```
0.15.216.3.10 size() template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
std::size_t smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::size (
) const [inline]
```

Returns the number of vertices.

```
0.15.216.3.11 toDot() template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
```

```
void smtrat::ForwardHyperGraph< VertexProperty, EdgeProperty, UniqueVertices >::toDot (
 const std::string & filename) const
Writes a representation of this graph to the given file.
```

#### 0.15.216.4 Friends And Related Function Documentation

```
0.15.216.4.1 operator<< template<typename VertexProperty , typename EdgeProperty , bool UniqueVertices = true>
template<typename VP , typename EP >
std::ostream& operator<< (
 std::ostream & os,
 const ForwardHyperGraph< VP, EP > & fhg) [friend]
Writes a textual representation of this graph to the given stream.
```

### 0.15.217 smtrat::FouMoModule< Settings > Class Template Reference

A module which applies the Fourier-Motzkin algorithm.

```
#include <FouMoModule.h>
```

#### Public Types

- **typedef Settings SettingsType**
- **enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }**

#### Public Member Functions

- **std::string moduleName () const**  
*The module has to take the given sub-formula of the received formula into account.*
- **FouMoModule (const ModuleInput \*\_formula, Conditionals &\_conditionals, Manager \*\_manager=NULL)**  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- **~FouMoModule ()**
- **bool addCore (ModuleInput::const\_iterator \_subformula)**  
*The module has to take the given sub-formula of the received formula into account.*
- **void removeCore (ModuleInput::const\_iterator \_subformula)**  
*Updates the current assignment into the model.*
- **void updateModel () const**  
*Updates the current assignment into the model.*
- **Answer checkCore ()**  
*Checks the received formula for consistency.*
- **bool inform (const FormulaT &\_constraint)**  
*Informs the module about the given constraint.*
- **void deinform (const FormulaT &\_constraint)**  
*The inverse of informing about a constraint.*
- **virtual void init ()**  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- **bool add (ModuleInput::const\_iterator \_subformula)**  
*The module has to take the given sub-formula of the received formula into account.*
- **virtual Answer check (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)**  
*Checks the received formula for consistency.*
- **virtual void remove (ModuleInput::const\_iterator \_subformula)**  
*Removes everything related to the given sub-formula of the received formula.*
- **virtual void updateAllModels ()**  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- **virtual std::list< std::vector< carl::Variable > > getModelEqualities () const**

*Partition the variables from the current model into equivalence classes according to their assigned value.*

- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`

*Sets this modules unique ID to identify itself.*

- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*

- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`
- `const std::vector< Model > & allModels () const`
- `const std::vector< FormulaSetT > & infeasibleSubsets () const`
- `const std::vector< Module * > & usedBackends () const`
- `const carl::FastSet< FormulaT > & constraintsToInform () const`
- `const carl::FastSet< FormulaT > & informedConstraints () const`
- `void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*

- `bool hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*

- `void clearLemmas ()`

*Deletes all yet found lemmas.*

- `const std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- `void receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*

- `const smrat::Conditionals & answerFound () const`
- `bool isPreprocessor () const`
- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*

- `void updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*

- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*

- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*

- `virtual std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*

- void `printReceivedFormula` (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of the received formula.*
- void `printPassedFormula` (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const  
*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector<`Branching`> `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector<`FormulaT`> `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair<`ModuleInput::iterator`, bool> `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)

- std::pair< [ModuleInput::iterator](#), bool > [removeOrigins](#) ([ModuleInput::iterator](#) \_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
- void [informBackends](#) (const [FormulaT](#) &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- virtual void [addConstraintToInform](#) (const [FormulaT](#) &\_constraint)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< [ModuleInput::iterator](#), bool > [addReceivedSubformulaToPassedFormula](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool [originInReceivedFormula](#) (const [FormulaT](#) &\_origin) const
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
 

*Finds the best origin from the given set of origins.*
- void [getInfeasibleSubsets](#) ()
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void [getBackendsModel](#) () const
 

*Get the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void [getBackendsAllModels](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
- virtual [Answer](#) [runBackends](#) (bool \_final, bool \_full, carl::Variable \_objective)
 

*Runs the backend solvers on the passed formula.*
- virtual [Answer](#) [runBackends](#) ()
 

*Runs the backend solvers on the passed formula.*
- virtual [ModuleInput::iterator](#) [eraseSubformulaFromPassedFormula](#) ([ModuleInput::iterator](#) \_subformula, bool \_ignoreOrigins=false)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void [clearPassedFormula](#) ()
 

*Clears the passed formula.*
- std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const
 

*Merges the two vectors of sets into the first one.*
- size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
 

*Determines the smallest origin in the given vector.*
- bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).*

- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector<`FormulaT`> &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set<`FormulaT`> & `getInformationRelevantFormulas` ()
 

*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
 

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector<`FormulaSetT`> `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector<`Model`> `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector<`TheoryPropagation`> `mTheoryPropagations`
- std::atomic<`Answer`> `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals` `mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector<`Module` \*> `mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector<`Module` \*> `mAllBackends`

*The backends of this module which have been used.*

- `std::vector< Lemma > mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- `carl::FastSet< FormulaT > mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.217.1 Detailed Description

```
template<typename Settings>
class smrat::FouMoModule< Settings >
```

A module which applies the Fourier-Motzkin algorithm.

### 0.15.217.2 Member Typedef Documentation

```
0.15.217.2.1 SettingsType template<typename Settings >
typedef Settings smrat::FouMoModule< Settings >::SettingsType
```

### 0.15.217.3 Member Enumeration Documentation

```
0.15.217.3.1 LemmaType enum smrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.217.4 Constructor & Destructor Documentation

```
0.15.217.4.1 FouMoModule() template<typename Settings >
smrat::FouMoModule< Settings >::FouMoModule (
 const ModuleInput * _formula,
 Conditionals & _conditionals,
 Manager * _manager = NULL)
```

```
0.15.217.4.2 ~FouMoModule() template<typename Settings >
smrat::FouMoModule< Settings >::~FouMoModule () [inline]
```

### 0.15.217.5 Member Function Documentation

**0.15.217.5.1 add()** `bool smtrat::Module::add (`  
    `ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.217.5.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`  
    `const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

#### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.217.5.3 addCore()** `template<typename Settings >`  
`bool smtrat::FouMoModule< Settings >::addCore (`  
    `ModuleInput::const_iterator _subformula ) [virtual]`

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.217.5.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula (`  
    `const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

```
0.15.217.5.5 addLemma() void smtrat::Module::addLemma (
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| _lemma            | The eduction/lemma to store.                                          |
| _lt               | The type of the lemma.                                                |
| _preferredFormula | Hint for the next decision, which formula should be assigned to true. |

```
0.15.217.5.6 addOrigin() void smtrat::Module::addOrigin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

```
0.15.217.5.7 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

```
0.15.217.5.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.217.5.9 addSubformulaToPassedFormula()** [2/3] std::pair<ModuleInput::iterator,bool> smrat→  
::Module::addSubformulaToPassedFormula ( const FormulaT & \_formula,  
const FormulaT & \_origin ) [inline], [protected], [inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                 |
| _origin  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.217.5.10 addSubformulaToPassedFormula()** [3/3] std::pair<ModuleInput::iterator,bool>  
smrat::Module::addSubformulaToPassedFormula ( const FormulaT & \_formula,  
const std::shared\_ptr<std::vector<FormulaT>> & \_origins ) [inline], [protected],  
[inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                                         |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                               |
| _origins | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.217.5.11 allModels()** const std::vector<Model>& smrat::Module::allModels () const [inline],  
[inherited]

#### Returns

All satisfying assignments, if existent.

**0.15.217.5.12 anAnswerFound()** bool smrat::Module::anAnswerFound () const [inline], [protected],  
[inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

**0.15.217.5.13 answerFound()** const `smtrat::Conditionals`& `smtrat::Module::answerFound` ( ) const [inline], [inherited]

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.217.5.14 backendsModel()** const `Model` & `smtrat::Module::backendsModel` ( ) const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.217.5.15 branchAt() [1/4]** bool `smtrat::Module::branchAt` ( carl::Variable::Arg `_var`, const `Rational` & `_value`, bool `_leftCaseWeak` = `true`, bool `_preferLeftCase` = `true`, bool `_useReceivedFormulaAsPremise` = `false`, const std::vector< `FormulaT` > & `_premise` = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.217.5.16 branchAt() [2/4]** bool `smtrat::Module::branchAt` ( carl::Variable::Arg `_var`, const `Rational` & `_value`, std::vector< `FormulaT` > && `_premise`, bool `_leftCaseWeak` = `true`, bool `_preferLeftCase` = `true`, bool `_useReceivedFormulaAsPremise` = `false` ) [inline], [protected], [inherited]

**0.15.217.5.17 branchAt() [3/4]** bool `smtrat::Module::branchAt` ( const `Poly` & `_polynomial`, bool `_integral`, const `Rational` & `_value`, bool `_leftCaseWeak` = `true`, bool `_preferLeftCase` = `true`, bool `_useReceivedFormulaAsPremise` = `false`, const std::vector< `FormulaT` > & `_premise` = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.217.5.18 branchAt() [4/4]** bool `smtrat::Module::branchAt` ( const `Poly` & `_polynomial`, bool `_integral`, const `Rational` & `_value`, std::vector< `FormulaT` > && `_premise`, bool `_leftCaseWeak` = `true`, bool `_preferLeftCase` = `true`, bool `_useReceivedFormulaAsPremise` = `false` ) [protected], [inherited]

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                          |                             |
|--------------------------|-----------------------------|
| <code>_polynomial</code> | The variable to branch for. |
|--------------------------|-----------------------------|

## Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

### 0.15.217.5.19 `check()`

```
Answer smrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

## Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

## Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::PModule](#).

### 0.15.217.5.20 `checkCore()`

```
template<typename Settings >
Answer smrat::FouMoModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

## Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

### 0.15.217.5.21 `checkInfSubsetForMinimality()`

```
void smrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

## Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.217.5.22 `checkModel()`** `unsigned smrat::Module::checkModel ( ) const [protected], [inherited]`

## Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.217.5.23 `cleanModel()`** `void smrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.217.5.24 `clearLemmas()`** `void smrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.217.5.25 `clearModel()`** `void smrat::Module::clearModel ( ) const [inline], [protected], [inherited]`  
Clears the assignment, if any was found.

**0.15.217.5.26 `clearModels()`** `void smrat::Module::clearModels ( ) const [inline], [protected], [inherited]`  
Clears all assignments, if any was found.

**0.15.217.5.27 `clearPassedFormula()`** `void smrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.217.5.28 `collectOrigins() [1/2]`** `void smrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

**0.15.217.5.29 `collectOrigins() [2/2]`** `void smrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.217.5.30 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations ( )`  
[inherited]

**0.15.217.5.31 `constraintsToInform()`** `const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]`

**Returns**

The constraints which the backends must be informed about.

**0.15.217.5.32 `currentlySatisfied()`** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.217.5.33 `currentlySatisfiedByBackend()`** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.217.5.34 `deinform()`** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.217.5.35 `deinformCore()`** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

## Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAbstractModule< Settings >](#), [smtrat::LRAbstractModule< LRASettingsICP >](#), and [smtrat::LRAbstractModule< LRASettings4 >](#).

**0.15.217.5.36 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (const std::vector< FormulaT > & origins) const [protected], [inherited]`

## Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

## Returns

The index of the smallest (regarding the size of the sets) element of `origins`

**0.15.217.5.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins = false) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

## Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

## Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.217.5.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel () const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.217.5.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (const std::vector< FormulaT > & _origins) const [protected], [inherited]`

## Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

Returns

**0.15.217.5.40 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.217.5.41 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.217.5.42 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.217.5.43 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.217.5.44 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.217.5.45 getBackendsModel()** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.217.5.46 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`  
Copies the infeasible subsets of the passed formula.

**0.15.217.5.47 getInfeasibleSubsets() [2/2]** `std::vector<FormulaSetT> smtrat::Module::getInfeasibleSubsets (const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.217.5.48 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas () [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.217.5.49 getModelEqualities()** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities () const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.217.5.50 getOrigins() [1/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.217.5.51 getOrigins() [2/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.217.5.52 getOrigins() [3/3]** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.217.5.53 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified( ) [virtual], [inherited]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.217.5.54 `hasLemmas()`** `bool smtrat::Module::hasLemmas( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

**0.15.217.5.55 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.217.5.56 `id()`** `std::size_t smtrat::Module::id( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.217.5.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.217.5.58 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.217.5.59 informBackends()** void smrat::Module::informBackends ( const FormulaT & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**0.15.217.5.60 informCore()** virtual bool smrat::Module::informCore (

const FormulaT & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.217.5.61 informedConstraints()** const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.217.5.62 init()** void smrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

**0.15.217.5.63 is\_minimizing()** bool smrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.217.5.64 `isLemmaLevel()`** `bool smtrat::Module::isLemmaLevel (``LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.217.5.65 `isPreprocessor()`** `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.217.5.66 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.217.5.67 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (``const std::vector< FormulaT > & _vecSetA,``const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.217.5.68 `model()`** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.217.5.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (``const Model & _modelA,``const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.217.5.70 `moduleName()`** template<typename Settings >  
`std::string smtrat::FouMoModule< Settings >::moduleName ( ) const [inline], [virtual]`

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.217.5.71 `objective()`** carl::Variable `smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.217.5.72 `originInReceivedFormula()`** bool `smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.217.5.73 `passedFormulaBegin()`** [ModuleInput::iterator](#) `smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.217.5.74 `passedFormulaEnd()`** [ModuleInput::iterator](#) `smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.217.5.75 `pPassedFormula()`** const [ModuleInput\\*](#) `smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.217.5.76 `pReceivedFormula()`** const [ModuleInput\\*](#) `smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.217.5.77 `print()`** void `smtrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.217.5.78 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.217.5.79 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.217.5.80 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.217.5.81 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.217.5.82 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.217.5.83 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.217.5.84 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline], [inherited]
```

Notifies that the received formulas has been checked.

```
0.15.217.5.85 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs< InfeasibleSubset (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.217.5.86 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.217.5.87 remove() void smtrat::Module::remove (
```

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

```
0.15.217.5.88 removeCore() template<typename Settings >
void smtrat::FouMoModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.217.5.89 `removeOrigin()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (`

`ModuleInput::iterator _formula,`  
`const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.217.5.90 `removeOrigins()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (`

`ModuleInput::iterator _formula,`  
`const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],`  
`[inherited]`

**0.15.217.5.91 `rPassedFormula()`** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const`  
[inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.217.5.92 `rReceivedFormula()`** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const`  
[inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.217.5.93 `runBackends() [1/2]`** `virtual Answer smtrat::Module::runBackends ( ) [inline],`  
[protected], [virtual], [inherited]

Reimplemented in [smtrat::PModule](#).

**0.15.217.5.94 `runBackends() [2/2]`** `Answer smtrat::Module::runBackends (`

`bool _final,`  
`bool _full,`  
`carl::Variable _objective ) [protected], [virtual], [inherited]`

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.217.5.95 setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]  
 Sets this modules unique ID to identify itself.

#### Parameters

|                                                 |                                    |
|-------------------------------------------------|------------------------------------|
| $\leftarrow$<br>$\overline{\leftarrow}$<br>$id$ | The id to set this module's id to. |
|-------------------------------------------------|------------------------------------|

**0.15.217.5.96 setThreadPriority()** void smtrat::Module::setThreadPriority ( thread\_priority \_threadPriority ) [inline], [inherited]  
 Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| $_threadPriority$ | The priority to set this module's thread priority to. |
|-------------------|-------------------------------------------------------|

**0.15.217.5.97 solverState()** Answer smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.217.5.98 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]  
 Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                      |                                                  |
|----------------------|--------------------------------------------------|
| $_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|----------------------|--------------------------------------------------|

**0.15.217.5.99 threadPriority()** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.217.5.100 updateAllModels()** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]  
 Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
 Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.217.5.101 updateLemmas()** void smtrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.217.5.102 updateModel()** template<typename Settings >  
void smtrat::FouMoModule< Settings >::updateModel () const [virtual]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.217.5.103 usedBackends()** const std::vector<[Module](#)\*>& smtrat::Module::usedBackends ()  
const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.217.6 Field Documentation

**0.15.217.6.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected],  
[inherited]

The backends of this module which have been used.

**0.15.217.6.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected],  
[inherited]

Stores all satisfying assignments.

**0.15.217.6.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer  
[protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.217.6.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform  
[protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.217.6.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.217.6.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.217.6.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smtrat::Module::mFirstSubformulaToPass  
[protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.217.6.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.217.6.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.217.6.10 mFullCheck** `bool smtrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.217.6.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`  
Stores the infeasible subsets.

**0.15.217.6.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.217.6.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`  
Stores the last branches in a cycle buffer.

**0.15.217.6.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.217.6.15 mModel** `Model smtrat::Module::mModel [mutable], [protected], [inherited]`  
Stores the assignment of the current satisfiable result, if existent.

**0.15.217.6.16 mModelComputed** `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`  
True, if the model has already been computed.

**0.15.217.6.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.217.6.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.217.6.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables [static], [inherited]`  
Reusable splitting variables.

**0.15.217.6.20 mpManager** `Manager* const smtrat::Module::mpManager [protected], [inherited]`  
A reference to the manager.

**0.15.217.6.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.217.6.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.217.6.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheory->Propagations [protected], [inherited]`

**0.15.217.6.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.217.6.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`  
Maps variables to the number of their occurrences.

## 0.15.218 smtrat::qe::fm::FourierMotzkinQE Class Reference

Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints.

```
#include <FourierMotzkinQE.h>
```

### Public Types

- using `FormulaPartition = std::vector< std::vector< FormulaT > >`

### Public Member Functions

- `FourierMotzkinQE (const FormulaT &qfree, const QEQuery &quantifiers)`
- `FormulaT eliminateQuantifiers ()`

#### 0.15.218.1 Detailed Description

Provides a simple implementation for Fourier Motzkin variable elimination for linear, existentially quantified constraints.

#### 0.15.218.2 Member Typedef Documentation

**0.15.218.2.1 FormulaPartition** using `smtrat::qe::fm::FourierMotzkinQE::FormulaPartition = std::vector< std::vector< FormulaT > >`

#### 0.15.218.3 Constructor & Destructor Documentation

```
0.15.218.3.1 FourierMotzkinQE() smtrat::qe::fm::FourierMotzkinQE::FourierMotzkinQE (
 const FormulaT & qfree,
 const QEQuery & quantifiers) [inline]
```

#### 0.15.218.4 Member Function Documentation

**0.15.218.4.1 eliminateQuantifiers()** FormulaT smtrat::qe::fm::FourierMotzkinQE::eliminateQuantifiers( )

### 0.15.219 smtrat::FPPModule< Settings > Class Template Reference

```
#include <FPPModule.h>
```

#### Public Types

- **typedef Settings SettingsType**
- **enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }**

#### Public Member Functions

- **FPPModule (const ModuleInput \*\_formula, Conditionals &\_conditionals, Manager \*\_manager=NULL)**
- **~FPPModule ()**
- **bool informCore (const FormulaT &\_constraint)**

*Informs the module about the given constraint.*
- **void init ()**

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- **bool addCore (ModuleInput::const\_iterator \_subformula)**

*The module has to take the given sub-formula of the received formula into account.*
- **void removeCore (ModuleInput::const\_iterator \_subformula)**

*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- **void updateModel () const**

*Updates the current assignment into the model.*
- **Answer checkCore ()**

*Checks the received formula for consistency.*
- **bool isPreprocessor () const**
- **bool appliedPreprocessing () const**
- **bool add (ModuleInput::const\_iterator \_subformula)**

*The module has to take the given sub-formula of the received formula into account.*
- **void remove (ModuleInput::const\_iterator \_subformula)**

*Removes everything related to the given sub-formula of the received formula.*
- **Answer check (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)**

*Checks the received formula for consistency.*
- **Answer runBackends (bool \_final, bool \_full, carl::Variable \_objective)**

*Runs the backend solvers on the passed formula.*
- **virtual Answer runBackends ()**
- **std::pair< bool, FormulaT > getReceivedFormulaSimplified ()**
- **bool inform (const FormulaT &\_constraint)**

*Informs the module about the given constraint.*
- **void deinform (const FormulaT &\_constraint)**

*The inverse of informing about a constraint.*
- **virtual void updateAllModels ()**

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*

- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned **currentlySatisfiedByBackend** (const **FormulaT** &\_formula) const
- virtual unsigned **currentlySatisfied** (const **FormulaT** &) const
- bool **receivedVariable** (carl::Variable::Arg \_var) const
- **Answer solverState** () const
- std::size\_t **id** () const
- void **setId** (std::size\_t \_id)  
*Sets this module's unique ID to identify itself.*
- **thread\_priority threadPriority** () const
- void **setThreadPriority** (**thread\_priority** \_threadPriority)  
*Sets the priority of this module to get a thread for running its check procedure.*
- const **ModuleInput** \* **pReceivedFormula** () const
- const **ModuleInput** & **rReceivedFormula** () const
- const **ModuleInput** \* **pPassedFormula** () const
- const **ModuleInput** & **rPassedFormula** () const
- const **Model** & **model** () const
- const std::vector< **Model** > & **allModels** () const
- const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
- const std::vector< **Module** \* > & **usedBackends** () const
- const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
- const carl::FastSet< **FormulaT** > & **informedConstraints** () const
- void **addLemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lt=Lemmatype::NORMAL, const **FormulaT** &\_preferredFormula=**FormulaT**(carl::FormalType::TRUE))  
*Stores a lemma being a valid formula.*
- bool **hasLemmas** ()  
*Checks whether this module or any of its backends provides any lemmas.*
- void **clearLemmas** ()  
*Deletes all yet found lemmas.*
- const std::vector< **Lemma** > & **lemmas** () const
- **ModuleInput**::const\_iterator **firstUncheckedReceivedSubformula** () const
- **ModuleInput**::const\_iterator **firstSubformulaToPass** () const
- void **receivedFormulaChecked** ()  
*Notifies that the received formulas has been checked.*
- const **smrat::Conditionals** & **answerFound** () const
- virtual std::string **moduleName** () const
- carl::Variable **objective** () const
- bool **is\_minimizing** () const
- void **excludeNotReceivedVariablesFromModel** () const  
*Excludes all variables from the current model, which do not occur in the received formula.*
- void **updateLemmas** ()  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- void **collectTheoryPropagations** ()
- void **collectOrigins** (const **FormulaT** &\_formula, **FormulasT** &\_origins) const  
*Collects the formulas in the given formula, which are part of the received formula.*
- void **collectOrigins** (const **FormulaT** &\_formula, **FormulaSetT** &\_origins) const
- bool **isValidInfeasibleSubset** () const
- void **checkInfeasibleSubsetForMinimality** (std::vector< **FormulaSetT** >::const\_iterator \_insubset, const std::string &\_filename="smaller\_muses", unsigned \_maxSizeDifference=1) const  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void **print** (const std::string &\_initiation="\*\*\*") const  
*Prints everything relevant of the solver.*

- void `printReceivedFormula` (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of the received formula.*
- void `printPassedFormula` (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const  
*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< `Branching` > `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< `FormulaT` > `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual void `deinformCore` (const `FormulaT` &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)

- void `informBackends` (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
 

*Get the infeasible subsets the given backend provides.*
- const `Model` & `backendsModel` () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel` () const
 

*Stores all models of a backend in the list of all models of this module.*
- void `getBackendsAllModels` () const
 

*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `probablyLooping` (const typename `Poly::PolyType` &\_branchingPolynomial, const `Rational` &\_branchingValue) const
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &←\_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*

- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &`formula`)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
 

*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)
 

*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &`_modelA`, const `Model` &`_modelB`)
 

*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals `mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`

*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput`::iterator `mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.219.1 Member Typedef Documentation

**0.15.219.1.1 SettingsType** `template<typename Settings >`  
`typedef Settings smrat::FPPModule< Settings >::SettingsType`

### 0.15.219.2 Member Enumeration Documentation

**0.15.219.2.1 LemmaType** `enum smrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.219.3 Constructor & Destructor Documentation

**0.15.219.3.1 FPPModule()** `template<typename Settings >`  
`smrat::FPPModule< Settings >::FPPModule (`  
`const ModuleInput * _formula,`  
`Conditionals & _conditionals,`  
`Manager * _manager = NULL )`

**0.15.219.3.2 ~FPPModule()** `template<typename Settings >`  
`smrat::FPPModule< Settings >::~FPPModule ( )`

### 0.15.219.4 Member Function Documentation

**0.15.219.4.1 add()** `bool smrat::PModule::add (`  
`ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.219.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`

```
const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.219.4.3 addCore()** `template<typename Settings >`

```
bool smtrat::FPPModule< Settings >::addCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.219.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula`

```
(
```

```
const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.219.4.5 addLemma()** `void smtrat::Module::addLemma (`

```
const FormulaT & _lemma,
```

```
const LemmaType & _lt = LemmaType::NORMAL,
```

```
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
```

```
[inherited]
```

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.219.4.6 addOrigin()** void smtrat::Module::addOrigin (

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.219.4.7 addReceivedSubformulaToPassedFormula()** std::pair<ModuleInput::iterator, bool>

```
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.219.4.8 addSubformulaToPassedFormula() [1/3]** std::pair<ModuleInput::iterator, bool> smtrat::Module::addSubformulaToPassedFormula (

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.219.4.9 addSubformulaToPassedFormula() [2/3]** std::pair<ModuleInput::iterator, bool> smtrat::Module::addSubformulaToPassedFormula (

```
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.219.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>**

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.219.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]****Returns**

All satisfying assignments, if existent.

**0.15.219.4.12 anAnswerFound() bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.219.4.13 answerFound() const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.219.4.14 appliedPreprocessing() bool smrat::PModule::appliedPreprocessing () const [inline], [inherited]****Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

---

**0.15.219.4.15 `backendsModel()`** const `Model` & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.219.4.16 `branchAt() [1/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.219.4.17 `branchAt() [2/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.219.4.18 `branchAt() [3/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.219.4.19 `branchAt() [4/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [protected], [inherited]

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                            |                                                                                                                                                                                                                |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>   | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>     | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>        | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>      | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code> | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |

## Parameters

|                                           |                                                                                              |
|-------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise. |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                |

**0.15.219.4.20 `check()`** `Answer smtrat::PModule::check (`

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

## Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

## Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.219.4.21 `checkCore()`** `template<typename Settings >`

```
Answer smtrat::FPPModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

## Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.219.4.22 `checkInfeasibleSubsetForMinimality()`** `void smtrat::Module::checkInfeasibleSubsetForMinimality (`

```
 std::vector< FormulaSet >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

## Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.219.4.23 checkModel()** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.219.4.24 cleanModel()** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.219.4.25 clearLemmas()** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.219.4.26 clearModel()** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.219.4.27 clearModels()** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.219.4.28 clearPassedFormula()** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.219.4.29 collectOrigins() [1/2]** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.219.4.30 collectOrigins() [2/2]** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.219.4.31 collectTheoryPropagations()** `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

**0.15.219.4.32 constraintsToInform()** const carl::FastSet<[FormulaT](#)>& smtrat::Module::constraintsToInform( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.219.4.33 currentlySatisfied()** virtual unsigned smtrat::Module::currentlySatisfied( const [FormulaT](#) & ) const [inline], [virtual], [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.219.4.34 currentlySatisfiedByBackend()** unsigned smtrat::Module::currentlySatisfiedByBackend( const [FormulaT](#) & \_formula ) const [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.219.4.35 deinform()** void smtrat::Module::deinform( const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.219.4.36 deinformCore()** virtual void smtrat::Module::deinformCore( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.219.4.37 determine\_smallest\_origin()** size\_t smtrat::Module::determine\_smallest\_origin( const std::vector< [FormulaT](#) > & origins ) const [protected], [inherited]

**Parameters**

|                |                              |
|----------------|------------------------------|
| <i>origins</i> | A vector of sets of origins. |
|----------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.219.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::erase<SubformulaFromPassedFormula (`

`ModuleInput::iterator _subformula,`  
`bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                       |                                                                                                              |
|-----------------------|--------------------------------------------------------------------------------------------------------------|
| <i>_subformula</i>    | The sub-formula to remove from the passed formula.                                                           |
| <i>_ignoreOrigins</i> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.219.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceived<VariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.219.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::find<BestOrigin (`

`const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

**0.15.219.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformula<ToPass ( ) const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.219.4.42 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

#### Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.219.4.43 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.219.4.44 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.219.4.45 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.219.4.46 getBackendsModel()** `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.219.4.47 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.219.4.48 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (`

`const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.219.4.49 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.219.4.50 getModelEqualities()** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.219.4.51 getOrigins() [1/3]** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulaSetT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.219.4.52 getOrigins() [2/3]** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulasT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.219.4.53 getOrigins() [3/3]** `const FormulaT& smtrat::Module::getOrigins (`

```
ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.219.4.54 getReceivedFormulaSimplified()** `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.219.4.55 hasLemmas()** `bool smtrat::Module::hasLemmas () [inline], [inherited]`  
 Checks whether this module or any of its backends provides any lemmas.

**0.15.219.4.56 hasValidInfeasibleSubset()** `bool smtrat::Module::hasValidInfeasibleSubset () const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.219.4.57 id()** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.219.4.58 infeasibleSubsets()** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.219.4.59 inform()** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.219.4.60 informBackends()** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.219.4.61 informCore()** `template<typename Settings > bool smtrat::FPPModule< Settings >::informCore (`

```
 const FormulaT & _constraint) [virtual]
```

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

`false`, if it can be easily decided whether the given constraint is inconsistent; `true`, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.219.4.62 informedConstraints()** `const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

#### Returns

The position of the first constraint of which no backend has been informed about.

**0.15.219.4.63 init()** `template<typename Settings > void smtrat::FPPModule< Settings >::init ( ) [virtual]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not and will not be called more than once and only before the first runBackends call.

Reimplemented from [smtrat::Module](#).

**0.15.219.4.64 is\_minimizing()** `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

**0.15.219.4.65 isLemmaLevel()** `bool smtrat::Module::isLemmaLevel (`

```
 LemmaLevel level) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.219.4.66 isPreprocessor()** `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`

#### Returns

`true`, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.219.4.67 lemmas()** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.219.4.68 merge()** `std::vector< FormulaT > smtrat::Module::merge ( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`  
Merges the two vectors of sets into the first one.  
 $\{\{a,b\},\{a,c\}\}$  and  $\{\{b,d\},\{b\}\} \rightarrow \{\{a,b,d\},\{a,b\},\{a,b,c,d\},\{a,b,c\}\}$

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.219.4.69 model()** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`

**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.219.4.70 modelsDisjoint()** `bool smtrat::Module::modelsDisjoint ( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`  
Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.219.4.71 moduleName()** `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

**Returns**

The name of the given module type as name.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SplitSOSModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LVEModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#),

smrat::ICPModule< ICPSettings1 >, smrat::GBModule< Settings >, smrat::FouMoModule< Settings >, smrat::EQPreprocessingModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::CubeLIAModule< Settings >, smrat::CSplitModule< Settings >, smrat::BVMModule< Settings >, and smrat::BEModule< Settings >.

**0.15.219.4.72 objective()** carl::Variable smrat::Module::objective ( ) const [inline], [inherited]

**0.15.219.4.73 originInReceivedFormula()** bool smrat::Module::originInReceivedFormula ( const FormulaT & \_origin ) const [protected], [inherited]

**0.15.219.4.74 passedFormulaBegin()** ModuleInput::iterator smrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.219.4.75 passedFormulaEnd()** ModuleInput::iterator smrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.219.4.76 pPassedFormula()** const ModuleInput\* smrat::Module::pPassedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.219.4.77 pReceivedFormula()** const ModuleInput\* smrat::Module::pReceivedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.219.4.78 print()** void smrat::Module::print ( const std::string & \_initiation = "\*\*\*" ) const [inherited]

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.219.4.79 printAllModels()** void smrat::Module::printAllModels ( std::ostream & \_out = std::cout ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.219.4.80 `printInfeasibleSubsets()`** `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.219.4.81 `printModel()`** `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.219.4.82 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.219.4.83 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.219.4.84 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.219.4.85 receivedFormulaChecked()** void smrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.219.4.86 receivedFormulasAsInfeasibleSubset()** void smrat::Module::receivedFormulasAsInfeasibleSubset (

`ModuleInput::const_iterator _subformula`) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

**0.15.219.4.87 receivedVariable()** bool smrat::Module::receivedVariable (

`carl::Variable::Arg _var`) const [inline], [inherited]

**0.15.219.4.88 remove()** void smrat::PModule::remove (

`ModuleInput::const_iterator _subformula`) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub formula of the received formula to remove. |
|--------------------|----------------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.219.4.89 removeCore()** template<typename Settings >

void smrat::FPPModule< Settings >::removeCore (

`ModuleInput::const_iterator _subformula`) [virtual]

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>_subformula</i> | The position of the subformula to remove. |
|--------------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.219.4.90 removeOrigin()** std::pair<[ModuleInput::iterator](#),bool> smrat::Module::removeOrigin (

`ModuleInput::iterator _formula,`

```
const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.219.4.91 removeOrigins()** std::pair<[ModuleInput](#)::iterator, bool> smtrat::Module::removeOrigins (

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.219.4.92 rPassedFormula()** const [ModuleInput](#)& smtrat::Module::rPassedFormula () const [inline], [inherited]

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.219.4.93 rReceivedFormula()** const [ModuleInput](#)& smtrat::Module::rReceivedFormula () const [inline], [inherited]

#### Returns

A reference to the formula passed to this module.

**0.15.219.4.94 runBackends() [1/2]** virtual [Answer](#) smtrat::PModule::runBackends () [inline], [virtual], [inherited]

Reimplemented from [smtrat::Module](#).

**0.15.219.4.95 runBackends() [2/2]** [Answer](#) smtrat::PModule::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.219.4.96 setId()** void smtrat::Module::setId (

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

**Parameters**

|                               |                                    |
|-------------------------------|------------------------------------|
| $\leftrightarrow$             | The id to set this module's id to. |
| $\underline{\leftrightarrow}$ |                                    |

**0.15.219.4.97 `setThreadPriority()`** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority )` [inline], [inherited]

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.219.4.98 `solverState()`** `Answer smtrat::Module::solverState ( ) const` [inline], [inherited]

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.219.4.99 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint )` [protected], [inherited]

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.219.4.100 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const` [inline], [inherited]

**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.219.4.101 `updateAllModels()`** `void smtrat::Module::updateAllModels ( )` [virtual], [inherited]

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.219.4.102 `updateLemmas()`** `void smtrat::Module::updateLemmas ( )` [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.219.4.103 updateModel()** template<typename Settings>  
void smrat::FPPModule<Settings>::updateModel() const [virtual]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smrat::Module](#).

**0.15.219.4.104 usedBackends()** const std::vector<[Module](#)\*>& smrat::Module::usedBackends() const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.219.5 Field Documentation

**0.15.219.5.1 mAllBackends** std::vector<[Module](#)\*> smrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

**0.15.219.5.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.219.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer [protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.219.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.219.5.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.219.5.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.219.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.219.5.8 mFirstUncheckedReceivedSubformula** [ModuleInput](#)::const\_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.219.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.219.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.219.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.219.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.219.5.13 mLastBranches** `std::vector<Branching>` `smtrat::Module::mLastBranches` = `std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.219.5.14 mLemmas** `std::vector<Lemma>` `smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.219.5.15 mModel** `Model` `smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.219.5.16 mModelComputed** `bool` `smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.219.5.17 mNumOfBranchVarsToStore** `std::size_t` `smtrat::Module::mNumOfBranchVarsToStore` = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.219.5.18 mObjectiveVariable** `carl::Variable` `smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.219.5.19 mOldSplittingVariables** `std::vector<FormulaT>` `smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.219.5.20 mpManager** `Manager*` `const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.219.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter`  
 [mutable], [protected], [inherited]  
 Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.219.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected],  
 [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.219.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheory←Propagations` [protected], [inherited]

**0.15.219.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends` [protected],  
 [inherited]  
 The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.219.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters`  
 [protected], [inherited]  
 Maps variables to the number of their occurrences.

## 0.15.220 smtrat::freelist< T > Class Template Reference

Implements a fixed-(at-runtime)-size allocator based on a free list embedded in the space of currently unused objects.

```
#include <alloc.h>
```

### Public Member Functions

- `freelist ()`
- `freelist (std::size_t chunk_size_)`
- `~freelist ()`
- template<typename... Args>  
`T & emplace (Args &&... args)`
- `void * alloc ()`  
*Allocate an object of size 1<<mSizeShift.*
- `void free (void *ptr) NOEXCEPT`  
*Freeing means just prepending the free'd element to the front of the freelist.*
- `void clear () NOEXCEPT`  
*Invalidate ALL entries allocated using this allocator.*

### 0.15.220.1 Detailed Description

```
template<typename T>
class smtrat::freelist< T >
```

Implements a fixed-(at-runtime)-size allocator based on a free list embedded in the space of currently unused objects.

Caution: Do not use this with types that need their destructor to be called, only for trivial types (like pointers, iterators, scalar values, ...). Otherwise, do not use the `clear()` method and call the destructor before `free()`.

### 0.15.220.2 Constructor & Destructor Documentation

**0.15.220.2.1 freelist() [1/2]** template<typename T >  
smtrat::freelist< T >::freelist ( ) [inline], [explicit]

**0.15.220.2.2 freelist() [2/2]** template<typename T >  
smtrat::freelist< T >::freelist ( std::size\_t chunk\_size\_ ) [inline], [explicit]

**0.15.220.2.3 ~freelist()** template<typename T >  
smtrat::freelist< T >::~freelist ( ) [inline]

### 0.15.220.3 Member Function Documentation

**0.15.220.3.1 alloc()** template<typename T >  
void\* smtrat::freelist< T >::alloc ( ) [inline]

Allocate an object of size  $1 \ll \text{mSizeShift}$ .

First, try to return the first entry on the freelist.

If that is empty, try forward-allocation (allocating the next element on the current page that has never been allocated before.)

Otherwise, go to the next page.

If there is no next page, allocate a new page with malloc. This page has twice the space of the previous page.

Start forward-allocation phase on the new page.

**0.15.220.3.2 clear()** template<typename T >  
void smtrat::freelist< T >::clear ( ) [inline]

Invalidate ALL entries allocated using this allocator.

Note that this does not call the destructor of any elements, and does not return any memory to the OS.

**0.15.220.3.3 emplace()** template<typename T >  
template<typename... Args>  
T& smtrat::freelist< T >::emplace ( Args &&... args ) [inline]

**0.15.220.3.4 free()** template<typename T >  
void smtrat::freelist< T >::free ( void \* ptr ) [inline]

Freeling means just prepending the free'd element to the front of the freelist.

## 0.15.221 smtrat::mcsat::FullParallelExplanation< Backends > Struct Template Reference

This explanation executes all given explanation parallel in multiple threads and waits until every single one has finished, returning the result of the first listed explanation.

```
#include <FullParallelExplanation.h>
```

### 0.15.221.1 Detailed Description

```
template<typename... Backends>
struct smtrat::mcsat::FullParallelExplanation< Backends >
```

This explanation executes all given explanation parallel in multiple threads and waits until every single one has finished, returning the result of the first listed explanation.

### 0.15.222 smrat::cad::FullSampleComparator< Iterator, Strategy > Struct Template Reference

```
#include <SampleComparator.h>
```

### 0.15.223 smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::T > Struct Template Reference

```
#include <SampleComparator.h>
```

### 0.15.224 smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Type > Struct Template Reference

```
#include <SampleComparator.h>
```

### 0.15.225 smrat::cad::FullSampleComparator< Iterator, FullSampleCompareStrategy::Value > Struct Template Reference

```
#include <SampleComparator.h>
```

## 0.15.226 smrat::parser::FunctionInstantiator Struct Reference

```
#include <FunctionInstantiator.h>
```

### Public Member Functions

- virtual ~FunctionInstantiator ()
- template<typename T>  
  bool convert (const std::vector< types::TermType > &from, std::vector< T > &to) const
- template<typename T>  
  bool convert (const std::vector< types::TermType > &from, std::vector< T > &to, TheoryError &errors) const
- virtual bool operator() (const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)  
  const

### 0.15.226.1 Constructor & Destructor Documentation

**0.15.226.1.1 ~FunctionInstantiator()** virtual smrat::parser::FunctionInstantiator::~FunctionInstantiator ( ) [inline], [virtual]

### 0.15.226.2 Member Function Documentation

**0.15.226.2.1 convert() [1/2]** template<typename T>  
bool smrat::parser::FunctionInstantiator::convert (  
  const std::vector< types::TermType > & from,  
  std::vector< T > & to) const [inline]

**0.15.226.2.2 convert() [2/2]** template<typename T>  
bool smrat::parser::FunctionInstantiator::convert (  
  const std::vector< types::TermType > & from,  
  std::vector< T > & to,  
  TheoryError & errors) const [inline]

---

```
0.15.226.2.3 operator() virtual bool smtrat::parser::FunctionInstantiator::operator() (
 const std::vector< types::TermType > &,
 types::TermType &,
 TheoryError & errors) const [inline], [virtual]
```

## 0.15.227 smtrat::GBModule< Settings > Class Template Reference

A solver module based on Groebner basis.

```
#include <GBModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `typedef Settings::Order Order`
- `typedef Settings::PolynomialWithReasons GBPolynomial`
- `typedef Settings::Multivariateideal Ideal`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
  - The module has to take the given sub-formula of the received formula into account.*
- `GBModule (const ModuleInput *const, Conditionals &, Manager *const =NULL)`
- `virtual ~GBModule ()`
- `bool addCore (ModuleInput::const_iterator _formula)`
  - Checks the received formula for consistency.*
- `virtual Answer checkCore ()`
  - Removes everything related to the given sub-formula of the received formula.*
- `void removeCore (ModuleInput::const_iterator _formula)`
  - Informs the module about the given constraint.*
- `void printStateHistory ()`
- `void printRewriteRules ()`
- `bool inform (const FormulaT &_constraint)`
  - The inverse of informing about a constraint.*
- `void deinform (const FormulaT &_constraint)`
  - Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `bool add (ModuleInput::const_iterator _subformula)`
  - Checks the received formula for consistency.*
- `virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`
  - Removes everything related to the given sub-formula of the received formula.*
- `virtual void remove (ModuleInput::const_iterator _subformula)`
  - Updates the model, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual void updateModel () const`
  - Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`
  - Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`

- std::size\_t `id () const`
- void `setId (std::size_t _id)`

*Sets this module's unique ID to identify itself.*
- `thread_priority threadPriority () const`
- void `setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput * pReceivedFormula () const`
- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const std::vector< `Model` > & `allModels () const`
- const std::vector< `FormulaSetT` > & `infeasibleSubsets () const`
- const std::vector< `Module` \* > & `usedBackends () const`
- const carl::FastSet< `FormulaT` > & `constraintsToInform () const`
- const carl::FastSet< `FormulaT` > & `informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `hasValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, `FormulaT` > `getReceivedFormulaSimplified ()`
- void `print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`

- Prints the infeasible subsets.
  - void **printModel** (std::ostream &\_out=std::cout) const
    - Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.
  - void **printAllModels** (std::ostream &\_out=std::cout)
    - Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

## Static Public Member Functions

- static void **freeSplittingVariable** (const **FormulaT** &\_splittingVariable)

## Static Public Attributes

- static size\_t **mNumOfBranchVarsToStore** = 5
  - The number of different variables to consider for a probable infinite loop of branchings.
- static std::vector< **Branching** > **mLastBranches** = std::vector<**Branching**>(mNumOfBranchVarsToStore, **Branching**(typename Poly::PolyType(), 0))
  - Stores the last branches in a cycle buffer.
- static size\_t **mFirstPosInLastBranches** = 0
  - The beginning of the cyclic buffer storing the last branches.
- static std::vector< **FormulaT** > **mOldSplittingVariables**
  - Reusable splitting variables.

## Protected Member Functions

- bool **constraintByGB** (carl::Relation cr)
- void **pushBacktrackPoint** (ModuleInput::const\_iterator btpoint)
- void **popBacktrackPoint** (ModuleInput::const\_iterator btpoint)
- bool **saveState** ()
- **FormulasT** **generateReasons** (const carl::BitVector &reasons)
- **FormulaSetT** **generateReasonSet** (const carl::BitVector &reasons)
- void **passGB** ()
- void **knownConstraintDeduction** (const std::list< std::pair< carl::BitVector, carl::BitVector > > &deductions)
- void **newConstraintDeduction** ()
- void **factorisedConstraintDeduction** (const std::list< **GBPolynomial** > &factorisation, const carl::BitVector &reasons)
- **GBPolynomial** **transformIntoEquality** (ModuleInput::const\_iterator constraint)
- **GBPolynomial** **callGroebnerToSDP** (const **Ideal** &gb)
- bool **iterativeVariableRewriting** ()
- bool **findTrivialFactorisations** ()
- void **processNewConstraint** (ModuleInput::const\_iterator \_formula)
- void **handleConstraintToGBQueue** (ModuleInput::const\_iterator \_formula)
- void **handleConstraintNotToGB** (ModuleInput::const\_iterator \_formula)
- void **removeReceivedFormulaFromNewInequalities** (ModuleInput::const\_iterator \_formula)
- void **removeSubformulaFromPassedFormula** (ModuleInput::iterator \_formula)
- **GBPolynomial** **rewriteVariable** (const **GBPolynomial** &, const carl::Variable &, const **TermT** &, const BitVector &)
- bool **validityCheck** ()
- virtual bool **informCore** (const **FormulaT** &\_constraint)
  - Informs the module about the given constraint.
- virtual void **deinformCore** (const **FormulaT** &\_constraint)
  - The inverse of informing about a constraint.
- bool **anAnswerFound** () const

- Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel () const`

*Clears the assignment, if any was found.*
  - void `clearModels () const`

*Clears all assignments, if any was found.*
  - void `cleanModel () const`

*Substitutes variable occurrences by its model value in the model values of other variables.*
  - `ModuleInput::iterator passedFormulaBegin ()`
  - `ModuleInput::iterator passedFormulaEnd ()`
  - void `addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
  - const `FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`

*Gets the origins of the passed formula at the given position.*
  - void `getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
  - void `getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
  - std::pair< `ModuleInput::iterator, bool` > `removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
  - std::pair< `ModuleInput::iterator, bool` > `removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
  - void `informBackends (const FormulaT &_constraint)`

*Informs all backends of this module about the given constraint.*
  - virtual void `addConstraintToInform (const FormulaT &_constraint)`

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
  - std::pair< `ModuleInput::iterator, bool` > `addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`

*Copies the given sub-formula of the received formula to the passed formula.*
  - bool `originInReceivedFormula (const FormulaT &_origin) const`
  - std::pair< `ModuleInput::iterator, bool` > `addSubformulaToPassedFormula (const FormulaT &_formula)`

*Adds the given formula to the passed formula with no origin.*
  - std::pair< `ModuleInput::iterator, bool` > `addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`

*Adds the given formula to the passed formula.*
  - std::pair< `ModuleInput::iterator, bool` > `addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`

*Adds the given formula to the passed formula.*
  - void `generateTrivialInfeasibleSubset ()`

*Stores the trivial infeasible subset being the set of received formulas.*
  - void `receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

*Stores an infeasible subset consisting only of the given received formula.*
  - std::vector< `FormulaT` >::const\_iterator `findBestOrigin (const std::vector< FormulaT > &_origins) const`
  - void `getInfeasibleSubsets ()`

*Copies the infeasible subsets of the passed formula.*
  - std::vector< `FormulaSetT` > `getInfeasibleSubsets (const Module &backend) const`

*Get the infeasible subsets the given backend provides.*
  - const `Model & backendsModel () const`

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - void `getBackendsModel () const`
  - void `getBackendsAllModels () const`

*Stores all models of a backend in the list of all models of this module.*
  - virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*

- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`
- std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const

*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename Poly::PolyType &\_branchingPolynomial, const `Rational` &\_branchingValue) const

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
- void `splitUnequalConstraint` (const `FormulaT` &)

*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel ()` const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)

*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas ()`

*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- Settings::Groebner `mBasis`

*The current Groebner basis.*
- InequalitiesTable< `Settings` > `mInequalities`

*The inequalities table for handling inequalities.*
- std::vector< `ModuleInput::const_iterator` > `mBacktrackPoints`

*The vector of backtrack points, which has pointers to received constraints.*
- std::list< `GBModuleState< Settings >` > `mStateHistory`

*Saves the relevant history to support backtracking.*
- bool `mRecalculateGB`

*After popping in the history, it might be necessary to recalculate. This flag indicates this.*
- std::list< typename InequalitiesTable< `Settings` >::Rows::iterator > `mNewInequalities`

*A list of inequalities which were added after the last consistency check.*

- **groebner::RewriteRules mRewriteRules**  
*The rewrite rules for the variables.*
- **std::map< ConstraintT, carl::Variable > mAdditionalVarMap**
- **FormulasT mGbEqualities**  
*A workaround to associate equalities in the passed formula originating from the gb (in contrast to those which originate from simplified formulae)*
- **std::vector< FormulaSetT > mInfeasibleSubsets**  
*Stores the infeasible subsets.*
- **Manager \*const mpManager**  
*A reference to the manager.*
- **Model mModel**  
*Stores the assignment of the current satisfiable result, if existent.*
- **std::vector< Model > mAllModels**  
*Stores all satisfying assignments.*
- **bool mModelComputed**  
*True, if the model has already been computed.*
- **bool mFinalCheck**  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- **bool mFullCheck**  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- **carl::Variable mObjectiveVariable**  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- **std::vector< TheoryPropagation > mTheoryPropagations**
- **std::atomic< Answer > mSolverState**  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- **std::atomic\_bool \* mBackendsFoundAnswer**  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- **Conditionals mFoundAnswer**  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- **std::vector< Module \* > mUsedBackends**  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- **std::vector< Module \* > mAllBackends**  
*The backends of this module which have been used.*
- **std::vector< Lemma > mLemmas**  
*Stores the lemmas being valid formulas this module or its backends made.*
- **ModuleInput::iterator mFirstSubformulaToPass**  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- **carl::FastSet< FormulaT > mConstraintsToInform**  
*Stores the constraints which the backends must be informed about.*
- **carl::FastSet< FormulaT > mInformedConstraints**  
*Stores the position of the first constraint of which no backend has been informed about.*
- **ModuleInput::const\_iterator mFirstUncheckedReceivedSubformula**  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- **unsigned mSmallerMusesCheckCounter**  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- **std::vector< std::size\_t > mVariableCounters**  
*Maps variables to the number of their occurrences.*

## Friends

- class **InequalitiesTable< Settings >**

### 0.15.227.1 Detailed Description

```
template<class Settings>
class smrat::GBModule< Settings >
```

A solver module based on Groebner basis.

Details can be found in my Bachelor Thesis "On Groebner Bases in SMT-Compliant Decision Procedures"

Author

Sebastian Junges

### 0.15.227.2 Member Typedef Documentation

#### 0.15.227.2.1 GBPolynomial template<class Settings >

```
typedef Settings::PolynomialWithReasons smrat::GBModule< Settings >::GBPolynomial
```

#### 0.15.227.2.2 Ideal template<class Settings >

```
typedef Settings::MultivariateIdeal smrat::GBModule< Settings >::Ideal
```

#### 0.15.227.2.3 Order template<class Settings >

```
typedef Settings::Order smrat::GBModule< Settings >::Order
```

#### 0.15.227.2.4 SettingsType template<class Settings >

```
typedef Settings smrat::GBModule< Settings >::SettingsType
```

### 0.15.227.3 Member Enumeration Documentation

#### 0.15.227.3.1 LemmaType enum smrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.227.4 Constructor & Destructor Documentation

#### 0.15.227.4.1 GBModule() template<class Settings >

```
smrat::GBModule< Settings >::GBModule (
 const ModuleInput * const ,
 Conditionals & ,
 Manager * const = NULL)
```

#### 0.15.227.4.2 ~GBModule() template<class Settings >

```
virtual smrat::GBModule< Settings >::~GBModule () [virtual]
```

### 0.15.227.5 Member Function Documentation

**0.15.227.5.1 add()** `bool smtrat::Module::add ( ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.227.5.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform ( const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

#### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.227.5.3 addCore()** `template<class Settings > bool smtrat::GBModule< Settings >::addCore ( ModuleInput::const_iterator formula ) [virtual]`

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.227.5.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula ( const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

---

**0.15.227.5.5 addLemma()** void smtrat::Module::addLemma (

```
const FormulaT & _lemma,
const LemmaType & _lt = LemmaType::NORMAL,
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| _lemma            | The eduction/lemma to store.                                          |
| _lt               | The type of the lemma.                                                |
| _preferredFormula | Hint for the next decision, which formula should be assigned to true. |

**0.15.227.5.6 addOrigin()** void smtrat::Module::addOrigin (

```
ModuleInput::iterator _formula,
const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

**0.15.227.5.7 addReceivedSubformulaToPassedFormula()** std::pair<ModuleInput::iterator,bool>  
smtrat::Module::addReceivedSubformulaToPassedFormula (

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

**0.15.227.5.8 addSubformulaToPassedFormula()** [1/3] std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (

```
const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.227.5.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                 |
| <i>_origin</i>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.227.5.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector<FormulaT> >& _origins) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.227.5.11 allModels() const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]
```

#### Returns

All satisfying assignments, if existent.

```
0.15.227.5.12 anAnswerFound() bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

---

**0.15.227.5.13 answerFound()** const `smtrat::Conditionals`& `smtrat::Module::answerFound` ( ) const [inline], [inherited]

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.227.5.14 backendsModel()** const `Model` & `smtrat::Module::backendsModel` ( ) const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.227.5.15 branchAt() [1/4]** bool `smtrat::Module::branchAt` ( carl::Variable::Arg `_var`, const `Rational` & `_value`, bool `_leftCaseWeak` = `true`, bool `_preferLeftCase` = `true`, bool `_useReceivedFormulaAsPremise` = `false`, const std::vector< `FormulaT` > & `_premise` = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.227.5.16 branchAt() [2/4]** bool `smtrat::Module::branchAt` ( carl::Variable::Arg `_var`, const `Rational` & `_value`, std::vector< `FormulaT` > && `_premise`, bool `_leftCaseWeak` = `true`, bool `_preferLeftCase` = `true`, bool `_useReceivedFormulaAsPremise` = `false` ) [inline], [protected], [inherited]

**0.15.227.5.17 branchAt() [3/4]** bool `smtrat::Module::branchAt` ( const `Poly` & `_polynomial`, bool `_integral`, const `Rational` & `_value`, bool `_leftCaseWeak` = `true`, bool `_preferLeftCase` = `true`, bool `_useReceivedFormulaAsPremise` = `false`, const std::vector< `FormulaT` > & `_premise` = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.227.5.18 branchAt() [4/4]** bool `smtrat::Module::branchAt` ( const `Poly` & `_polynomial`, bool `_integral`, const `Rational` & `_value`, std::vector< `FormulaT` > && `_premise`, bool `_leftCaseWeak` = `true`, bool `_preferLeftCase` = `true`, bool `_useReceivedFormulaAsPremise` = `false` ) [protected], [inherited]

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                          |                             |
|--------------------------|-----------------------------|
| <code>_polynomial</code> | The variable to branch for. |
|--------------------------|-----------------------------|

## Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

**0.15.227.5.19 `callGroebnerToSDP()`** template<class Settings >  
`GBPolynomial smtrat::GBModule< Settings >::callGroebnerToSDP (`  
`const Ideal & gb ) [protected]`

**0.15.227.5.20 `check()`** `Answer smtrat::Module::check (`

```
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

## Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

## Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.227.5.21 `checkCore()`** template<class Settings >  
`virtual Answer smtrat::GBModule< Settings >::checkCore ( ) [virtual]`

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

## Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.227.5.22 checkInfSubsetForMinimality() void smrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.227.5.23 checkModel() unsigned smrat::Module::checkModel () const [protected], [inherited]
```

#### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

```
0.15.227.5.24 cleanModel() void smrat::Module::cleanModel () const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

```
0.15.227.5.25 clearLemmas() void smrat::Module::clearLemmas () [inline], [inherited]
```

Deletes all yet found lemmas.

```
0.15.227.5.26 clearModel() void smrat::Module::clearModel () const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

```
0.15.227.5.27 clearModels() void smrat::Module::clearModels () const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

```
0.15.227.5.28 clearPassedFormula() void smrat::Module::clearPassedFormula () [protected], [inherited]
```

```
0.15.227.5.29 collectOrigins() [1/2] void smrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

```
0.15.227.5.30 collectOrigins() [2/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

```
0.15.227.5.31 collectTheoryPropagations() void smtrat::Module::collectTheoryPropagations () [inherited]
```

```
0.15.227.5.32 constraintByGB() template<class Settings >
bool smtrat::GBModule< Settings >::constraintByGB (
 carl::Relation cr) [protected]
```

```
0.15.227.5.33 constraintsToInform() const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]
```

#### Returns

The constraints which the backends must be informed about.

```
0.15.227.5.34 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAbstractModule< Settings >](#), [smtrat::LRAbstractModule< LRASettingsICP >](#), and [smtrat::LRAbstractModule< LRASettingsCP >](#)

```
0.15.227.5.35 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.227.5.36 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.227.5.37 `deinformCore()`** `virtual void smtrat::Module::deinformCore (`

```
 const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.227.5.38 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (`

```
 const std::vector< FormulaT > & origins) const [protected], [inherited]
```

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.227.5.39 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

```
 ModuleInput::iterator _subformula,
```

```
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.227.5.40 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

```
0.15.227.5.41 factorisedConstraintDeduction() template<class Settings >
void smtrat::GBModule< Settings >::factorisedConstraintDeduction (
 const std::list< GBPolynomial > & factorisation,
 const carl::BitVector & reasons) [protected]
```

```
0.15.227.5.42 findBestOrigin() std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (
 const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

#### Parameters

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

#### Returns

```
0.15.227.5.43 findTrivialFactorisations() template<class Settings >
bool smtrat::GBModule< Settings >::findTrivialFactorisations () [protected]
```

```
0.15.227.5.44 firstSubformulaToPass() ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]
```

#### Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

```
0.15.227.5.45 firstUncheckedReceivedSubformula() ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]
```

#### Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

```
0.15.227.5.46 freeSplittingVariable() static void smtrat::Module::freeSplittingVariable (
 const FormulaT & _splittingVariable) [inline], [static], [inherited]
```

```
0.15.227.5.47 generateReasons() template<class Settings >
FormulasT smtrat::GBModule< Settings >::generateReasons (
 const carl::BitVector & reasons) [protected]
```

```
0.15.227.5.48 generateReasonSet() template<class Settings >
FormulaSetT smtrat::GBModule< Settings >::generateReasonSet (
 const carl::BitVector & reasons) [protected]
```

```
0.15.227.5.49 generateTrivialInfeasibleSubset() void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

**0.15.227.5.50 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels () const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.227.5.51 getBackendsModel()** void smtrat::Module::getBackendsModel () const [protected], [inherited]

**0.15.227.5.52 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.227.5.53 getInfeasibleSubsets() [2/2]** std::vector< [FormulaSetT](#) > smtrat::Module::getInfeasibleSubsets (

const [Module](#) & [\\_backend](#) ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <a href="#">_backend</a> | The backend from which to obtain the infeasible subsets. |
|--------------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.227.5.54 getInformationRelevantFormulas()** const std::set< [FormulaT](#) > & smtrat::Module::getInformationRelevantFormulas () [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.227.5.55 getModelEqualities()** std::list< std::vector< [carl::Variable](#) > > smtrat::Module::getModelEqualities () const [virtual], [inherited]  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.227.5.56 getOrigins() [1/3]** void smtrat::Module::getOrigins (

const [FormulaT](#) & [\\_formula](#),

[FormulaSetT](#) & [\\_origins](#) ) const [inline], [protected], [inherited]

#### Parameters

|                          |  |
|--------------------------|--|
| <a href="#">_formula</a> |  |
| <a href="#">_origins</a> |  |

```
0.15.227.5.57 getOrigins() [2/3] void smtrat::Module::getOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inline], [protected], [inherited]
```

**Parameters**

|                 |  |
|-----------------|--|
| <u>_formula</u> |  |
| <u>_origins</u> |  |

```
0.15.227.5.58 getOrigins() [3/3] const FormulaT& smtrat::Module::getOrigins (
 ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <u>_formula</u> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

```
0.15.227.5.59 getReceivedFormulaSimplified() std::pair< bool, FormulaT > smtrat::Module::get<-
ReceivedFormulaSimplified () [virtual], [inherited]
```

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

```
0.15.227.5.60 handleConstraintNotToGB() template<class Settings >
void smtrat::GBModule< Settings >::handleConstraintNotToGB (
 ModuleInput::const_iterator _formula) [protected]
```

```
0.15.227.5.61 handleConstraintToGBQueue() template<class Settings >
void smtrat::GBModule< Settings >::handleConstraintToGBQueue (
 ModuleInput::const_iterator _formula) [protected]
```

```
0.15.227.5.62 hasLemmas() bool smtrat::Module::hasLemmas () [inline], [inherited]
```

Checks whether this module or any of its backends provides any lemmas.

```
0.15.227.5.63 hasValidInfeasibleSubset() bool smtrat::Module::hasValidInfeasibleSubset () const
[inherited]
```

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.227.5.64 `id()`** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.227.5.65 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.227.5.66 `inform()`** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.227.5.67 `informBackends()`** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.227.5.68 `informCore()`** `virtual bool smtrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.227.5.69 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.227.5.70 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.227.5.71 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.227.5.72 isLemmaLevel()** `bool smrat::Module::isLemmaLevel(`

`LemmaLevel level) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.227.5.73 isPreprocessor()** `bool smrat::Module::isPreprocessor( ) const [inline], [inherited]`

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.227.5.74 iterativeVariableRewriting()** `template<class Settings >`

`bool smrat::GBModule< Settings >::iterativeVariableRewriting( ) [protected]`

**0.15.227.5.75 knownConstraintDeduction()** `template<class Settings >`

`void smrat::GBModule< Settings >::knownConstraintDeduction(`  
`const std::list< std::pair< carl::BitVector, carl::BitVector > > & deductions )`  
`[protected]`

**0.15.227.5.76 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.227.5.77 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & \_vecSetA, const std::vector< [FormulaT](#) > & \_vecSetB ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.227.5.78 model()** const [Model](#)& smtrat::Module::model ( ) const [inline], [inherited]

**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.227.5.79 modelsDisjoint()** bool smtrat::Module::modelsDisjoint ( const [Model](#) & `_modelA`, const [Model](#) & `_modelB` ) [static], [protected], [inherited]

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.227.5.80 moduleName()** template<class Settings > std::string smtrat::GBModule< [Settings](#) >::moduleName ( ) const [inline], [virtual]

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

```
0.15.227.5.81 newConstraintDeduction() template<class Settings >
void smtrat::GBModule< Settings >::newConstraintDeduction () [protected]
```

**0.15.227.5.82 objective()** carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]

```
0.15.227.5.83 originInReceivedFormula() bool smtrat::Module::originInReceivedFormula (
 const FormulaT & _origin) const [protected], [inherited]
```

```
0.15.227.5.84 passedFormulaBegin() ModuleInput::iterator smtrat::Module::passedFormulaBegin (
) [inline], [protected], [inherited]
```

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

```
0.15.227.5.85 passedFormulaEnd() ModuleInput::iterator smtrat::Module::passedFormulaEnd ()
[inline], [protected], [inherited]
```

**Returns**

An iterator to the end of the passed formula.

```
0.15.227.5.86 passGB() template<class Settings >
void smtrat::GBModule< Settings >::passGB () [protected]
```

```
0.15.227.5.87 popBacktrackPoint() template<class Settings >
void smtrat::GBModule< Settings >::popBacktrackPoint (
 ModuleInput::const_iterator btpoint) [protected]
```

```
0.15.227.5.88 pPassedFormula() const ModuleInput* smtrat::Module::pPassedFormula () const
[inline], [inherited]
```

**Returns**

A pointer to the formula passed to the backends of this module.

```
0.15.227.5.89 pReceivedFormula() const ModuleInput* smtrat::Module::pReceivedFormula () const
[inline], [inherited]
```

**Returns**

A pointer to the formula passed to this module.

```
0.15.227.5.90 print() void smtrat::Module::print (
 const std::string & _initiation = "***") const [inherited]
```

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.227.5.91 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.227.5.92 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.227.5.93 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.227.5.94 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.227.5.95 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.227.5.96 printRewriteRules()** template<class Settings >  
void smtrat::GBModule< Settings >::printRewriteRules ( )

**0.15.227.5.97 printStateHistory()** template<class Settings >  
void smtrat::GBModule< Settings >::printStateHistory ( )

**0.15.227.5.98 probablyLooping()** bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & \_branchingPolynomial, const Rational & \_branchingValue ) const [protected], [inherited]

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.227.5.99 processNewConstraint()** template<class Settings >  
void smtrat::GBModule< Settings >::processNewConstraint ( ModuleInput::const\_iterator \_formula ) [protected]

**0.15.227.5.100 pushBacktrackPoint()** template<class Settings >  
void smtrat::GBModule< Settings >::pushBacktrackPoint ( ModuleInput::const\_iterator btpoint ) [protected]

**0.15.227.5.101 receivedFormulaChecked()** void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.227.5.102 receivedFormulasAsInfeasibleSubset()** void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const\_iterator \_subformula ) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

**0.15.227.5.103 receivedVariable()** bool smtrat::Module::receivedVariable ( carl::Variable::Arg \_var ) const [inline], [inherited]

**0.15.227.5.104 remove()** void smtrat::Module::remove ( ModuleInput::const\_iterator \_subformula ) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.227.5.105 `removeCore()`** template<class Settings >  
`void smtrat::GBModule< Settings >::removeCore (`  
`ModuleInput::const_iterator formula ) [virtual]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.227.5.106 `removeOrigin()`** std::pair<`ModuleInput::iterator`,`bool`> `smtrat::Module::removeOrigin (`  
`ModuleInput::iterator _formula,`  
`const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.227.5.107 `removeOrigins()`** std::pair<`ModuleInput::iterator`,`bool`> `smtrat::Module::removeOrigins (`  
`ModuleInput::iterator _formula,`  
`const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

**0.15.227.5.108 `removeReceivedFormulaFromNewInequalities()`** template<class Settings >  
`void smtrat::GBModule< Settings >::removeReceivedFormulaFromNewInequalities (`  
`ModuleInput::const_iterator _formula ) [protected]`

**0.15.227.5.109 `removeSubformulaFromPassedFormula()`** template<class Settings >  
`void smtrat::GBModule< Settings >::removeSubformulaFromPassedFormula (`  
`ModuleInput::iterator _formula ) [protected]`

**0.15.227.5.110 `rewriteVariable()`** template<class Settings >  
`GBPolynomial smtrat::GBModule< Settings >::rewriteVariable (`  
`const GBPolynomial & ,`  
`const carl::Variable & ,`  
`const TermT & ,`  
`const BitVector & ) [inline], [protected]`

**0.15.227.5.111 `rPassedFormula()`** const `ModuleInput&` `smtrat::Module::rPassedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.227.5.112 rReceivedFormula()** const [ModuleInput](#)& smtrat::Module::rReceivedFormula () const  
[inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.227.5.113 runBackends() [1/2]** virtual [Answer](#) smtrat::Module::runBackends () [inline],  
[protected], [virtual], [inherited]  
Reimplemented in [smtrat::PModule](#).

**0.15.227.5.114 runBackends() [2/2]** [Answer](#) smtrat::Module::runBackends (  
    bool \_final,  
    bool \_full,  
    carl::Variable \_objective ) [protected], [virtual], [inherited]

Runs the backend solvers on the passed formula.

**Parameters**

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.227.5.115 saveState()** template<class Settings >  
bool smtrat::GBModule< Settings >::saveState () [protected]

**0.15.227.5.116 setId()** void smtrat::Module::setId (  
    std::size\_t \_id ) [inline], [inherited]

Sets this modules unique ID to identify itself.

**Parameters**

|                             |                                    |
|-----------------------------|------------------------------------|
| $\leftrightarrow$<br>$\_id$ | The id to set this module's id to. |
|-----------------------------|------------------------------------|

**0.15.227.5.117 setThreadPriority()** void smtrat::Module::setThreadPriority (  
    [thread\\_priority](#) \_threadPriority ) [inline], [inherited]

Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

### 0.15.227.5.118 `solverState()` Answer smtrat::Module::solverState () const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

### 0.15.227.5.119 `splitUnequalConstraint()` void smtrat::Module::splitUnequalConstraint (const FormulaT & `_unequalConstraint`) [protected], [inherited]

Adds the following lemmas for the given constraint  $p \neq 0$ .

$$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

### 0.15.227.5.120 `threadPriority()` thread\_priority smtrat::Module::threadPriority () const [inline], [inherited]

#### Returns

The priority of this module to get a thread for running its check procedure.

### 0.15.227.5.121 `transformIntoEquality()` template<class Settings > GBPolynomial smtrat::GBModule< Settings >::transformIntoEquality (ModuleInput::const\_iterator constraint) [protected]

### 0.15.227.5.122 `updateAllModels()` void smtrat::Module::updateAllModels () [virtual], [inherited]

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in [smtrat::SATModule< Settings >](#).

### 0.15.227.5.123 `updateLemmas()` void smtrat::Module::updateLemmas () [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

### 0.15.227.5.124 `updateModel()` void smtrat::Module::updateModel () const [virtual], [inherited]

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::PModule](#), [smtrat::VSModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#),

`smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::MCBModule< Settings >, smrat::LVEModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntEqModule< Settings >, smrat::IntBlastModule< Settings >, smrat::IncWidthModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, smrat::ICPModule< ICPSettings1 >, smrat::GBPPModule< Settings >, smrat::FPPModule< Settings >, smrat::FouMoModule< Settings >, smrat::ESModule< Settings >, smrat::EQPreprocessingModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::CubeLIAModule< Settings >, smrat::CSplitModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.227.5.125 `usedBackends()`** `const std::vector<Module*>& smrat::Module::usedBackends( )`  
const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.227.5.126 `validityCheck()`** `template<class Settings >`  
`bool smrat::GBModule< Settings >::validityCheck( ) [protected]`

## 0.15.227.6 Friends And Related Function Documentation

**0.15.227.6.1 `InequalitiesTable< Settings >`** `template<class Settings >`  
friend class `InequalitiesTable< Settings >` [friend]

## 0.15.227.7 Field Documentation

**0.15.227.7.1 `mAdditionalVarMap`** `template<class Settings >`  
`std::map<ConstraintT, carl::Variable> smrat::GBModule< Settings >::mAdditionalVarMap [protected]`

**0.15.227.7.2 `mAllBackends`** `std::vector<Module*> smrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.227.7.3 `mAllModels`** `std::vector<Model> smrat::Module::mAllModels [mutable], [protected], [inherited]`  
Stores all satisfying assignments.

**0.15.227.7.4 `mBackendsFoundAnswer`** `std::atomic_bool* smrat::Module::mBackendsFoundAnswer [protected], [inherited]`  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.227.7.5 `mBacktrackPoints`** `template<class Settings >`  
`std::vector<ModuleInput::const_iterator> smrat::GBModule< Settings >::mBacktrackPoints [protected]`  
The vector of backtrack points, which has pointers to received constraints.

**0.15.227.7.6 mBasis** template<class Settings >  
Settings::Groebner smtrat::GBModule< Settings >::mBasis [protected]  
The current Groebner basis.

**0.15.227.7.7 mConstraintsToInform** carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform  
[protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.227.7.8 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.227.7.9 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.227.7.10 mFirstSubformulaToPass** ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.227.7.11 mFirstUncheckedReceivedSubformula** ModuleInput::const\_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.227.7.12 mFoundAnswer** Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.227.7.13 mFullCheck** bool smtrat::Module::mFullCheck [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.227.7.14 mGbEqualities** template<class Settings >  
FormulasT smtrat::GBModule< Settings >::mGbEqualities [protected]  
A workaround to associate equalities in the passed formula originating from the gb (in contrast to those which originate from simplified formulae)

**0.15.227.7.15 mInequalities** template<class Settings >  
InequalitiesTable<Settings> smtrat::GBModule< Settings >::mInequalities [protected]  
The inequalities table for handling inequalities.

**0.15.227.7.16 mInfeasibleSubsets** std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]  
Stores the infeasible subsets.

**0.15.227.7.17 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.227.7.18 mLastBranches** `std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.227.7.19 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.227.7.20 mModel Model** `smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.227.7.21 mModelComputed** `bool smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.227.7.22 mNewInequalities** `template<class Settings > std::list<typename InequalitiesTable<Settings>::Rows::iterator> smtrat::GBModule< Settings >::mNewInequalities` [protected]  
A list of inequalities which were added after the last consistency check.

**0.15.227.7.23 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.227.7.24 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.227.7.25 mOldSplittingVariables** `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.227.7.26 mpManager Manager\*** `const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.227.7.27 mRecalculateGB** `template<class Settings > bool smtrat::GBModule< Settings >::mRecalculateGB` [protected]  
After popping in the history, it might be necessary to recalculate. This flag indicates this.

**0.15.227.7.28 mRewriteRules** template<class Settings >  
`groebner::RewriteRules smrat::GBModule< Settings >::mRewriteRules [protected]`  
The rewrite rules for the variables.

**0.15.227.7.29 mSmallerMusesCheckCounter** unsigned smrat::Module::mSmallerMusesCheckCounter  
[mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.227.7.30 mSolverState** std::atomic<[Answer](#)> smrat::Module::mSolverState [protected],  
[inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.227.7.31 mStateHistory** template<class Settings >  
std::list<[GBModuleState<Settings>](#) > smrat::GBModule< Settings >::mStateHistory [protected]  
Saves the relevant history to support backtracking.

**0.15.227.7.32 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smrat::Module::mTheory->  
Propagations [protected], [inherited]

**0.15.227.7.33 mUsedBackends** std::vector<[Module\\*](#)> smrat::Module::mUsedBackends [protected],  
[inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.227.7.34 mVariableCounters** std::vector<std::size\_t> smrat::Module::mVariableCounters  
[protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.228 smrat::GBModuleState< Settings > Class Template Reference

A class to save the current state of a [GBModule](#).

```
#include <GBModuleState.h>
```

### Public Member Functions

- [GBModuleState \(\)](#)
- [GBModuleState \(const typename Settings::Groebner &basisCalculation, const std::map< carl::Variable, std::pair< TermT, carl::BitVector > > &rewrites\)](#)
- const Settings::Groebner & [getBasis \(\) const](#)
- const std::map< carl::Variable, std::pair< TermT, carl::BitVector > > & [getRewriteRules \(\) const](#)

### Protected Attributes

- const Settings::Groebner [mBasis](#)  
*The state of the basis.*
- const std::map< carl::Variable, std::pair< TermT, carl::BitVector > > [mRewrites](#)

### 0.15.228.1 Detailed Description

```
template<typename Settings>
class smrat::GBModuleState< Settings >
```

A class to save the current state of a [GBModule](#).  
Used for backtracking-support

### 0.15.228.2 Constructor & Destructor Documentation

**0.15.228.2.1 GBModuleState() [1/2]** template<typename Settings >  
`smrat::GBModuleState< Settings >:::GBModuleState ( ) [inline]`

**0.15.228.2.2 GBModuleState() [2/2]** template<typename Settings >  
`smrat::GBModuleState< Settings >:::GBModuleState (`  
    `const typename Settings::Groebner & basisCalculation,`  
    `const std::map< carl::Variable, std::pair< TermT, carl::BitVector > > & rewrites`  
`) [inline]`

### 0.15.228.3 Member Function Documentation

**0.15.228.3.1 getBasis()** template<typename Settings >  
`const Settings::Groebner& smrat::GBModuleState< Settings >::getBasis ( ) const [inline]`

**0.15.228.3.2 getRewriteRules()** template<typename Settings >  
`const std::map<carl::Variable, std::pair<TermT, carl::BitVector> >& smrat::GBModuleState< Settings >::getRewriteRules ( ) const [inline]`

### 0.15.228.4 Field Documentation

**0.15.228.4.1 mBasis** template<typename Settings >  
`const Settings::Groebner smrat::GBModuleState< Settings >::mBasis [protected]`  
The state of the basis.

**0.15.228.4.2 mRewrites** template<typename Settings >  
`const std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smrat::GBModuleState< Settings >::mRewrites [protected]`

## 0.15.229 smrat::GBPPModule< Settings > Class Template Reference

```
#include <GBPPModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0, PERMANENT = 1 }`

## Public Member Functions

- `GBPPModule` (const `ModuleInput` \*`_formula`, `Conditionals` &`_conditionals`, `Manager` \*`_manager=nullptr`)
- `~GBPPModule` ()
- void `updateModel` () const
 

*Updates the current assignment into the model.*
- `Answer checkCore` ()
 

*Checks the received formula for consistency.*
- bool `isPreprocessor` () const
- bool `appliedPreprocessing` () const
- bool `add` (`ModuleInput::const_iterator` `_subformula`)
 

*The module has to take the given sub-formula of the received formula into account.*
- void `remove` (`ModuleInput::const_iterator` `_subformula`)
 

*Removes everything related to the given sub-formula of the received formula.*
- `Answer check` (bool `_final=false`, bool `_full=true`, `carl::Variable` `_objective=carl::Variable::NO_VARIABLE`)
 

*Checks the received formula for consistency.*
- `Answer runBackends` (bool `_final`, bool `_full`, `carl::Variable` `_objective`)
 

*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends` ()
- std::pair< bool, `FormulaT` > `getReceivedFormulaSimplified` ()
 

*Informs the module about the given constraint.*
- void `deinform` (const `FormulaT` &`_constraint`)
 

*The inverse of informing about a constraint.*
- virtual void `init` ()
 

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- virtual void `updateAllModels` ()
 

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< `carl::Variable` > > `getModelEqualities` () const
 

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned `currentlySatisfiedByBackend` (const `FormulaT` &`_formula`) const
- virtual unsigned `currentlySatisfied` (const `FormulaT` &) const
- bool `receivedVariable` (`carl::Variable::Arg` `_var`) const
- `Answer solverState` () const
- std::size\_t `id` () const
- void `setId` (std::size\_t `_id`)
 

*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority` () const
- void `setThreadPriority` (`thread_priority` `_threadPriority`)
 

*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput` \* `pReceivedFormula` () const
- const `ModuleInput` & `rReceivedFormula` () const
- const `ModuleInput` \* `pPassedFormula` () const
- const `ModuleInput` & `rPassedFormula` () const
- const `Model` & `model` () const
- const std::vector< `Model` > & `allModels` () const
- const std::vector< `FormulaSetT` > & `infeasibleSubsets` () const
- const std::vector< `Module` \* > & `usedBackends` () const
- const `carl::FastSet< FormulaT >` & `constraintsToInform` () const
- const `carl::FastSet< FormulaT >` & `informedConstraints` () const
- void `addLemma` (const `FormulaT` &`_lemma`, const `LemmaType` &`_lt=LemmaType::NORMAL`, const `FormulaT` &`_preferredFormula=FormulaT(carl::FormulaType::TRUE)`)
 

*Stores a lemma being a valid formula.*

- `bool hasLemmas ()`  
*Checks whether this module or any of its backends provides any lemmas.*
- `void clearLemmas ()`  
*Deletes all yet found lemmas.*
- `const std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- `void receivedFormulaChecked ()`  
*Notifies that the received formulas has been checked.*
- `const smrat::Conditionals & answerFound () const`
- `virtual std::string moduleName () const`
- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`  
*Excludes all variables from the current model, which do not occur in the received formula.*
- `void updateLemmas ()`  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
*Collects the formulas in the given formula, which are part of the received formula.*
- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- `void print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- `void printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- `void printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- `void printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*
- `void printModel (std::ostream &_out=std::cout) const`  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- `void printAllModels (std::ostream &_out=std::cout)`  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- `static void freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- `static size_t mNumOfBranchVarsToStore = 5`  
*The number of different variables to consider for a probable infinite loop of branchings.*
- `static std::vector< Branching > mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`  
*Stores the last branches in a cycle buffer.*
- `static size_t mFirstPosInLastBranches = 0`  
*The beginning of the cyclic buffer storing the last branches.*
- `static std::vector< FormulaT > mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)
 

*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)
 

*The inverse of informing about a constraint.*
- virtual bool `addCore` (`ModuleInput::const_iterator` formula)
 

*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (`ModuleInput::const_iterator` formula)
 

*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const
 

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const
 

*Clears the assignment, if any was found.*
- void `clearModels` () const
 

*Clears all assignments, if any was found.*
- void `cleanModel` () const
 

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
 

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const
 

*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()

- Copies the infeasible subsets of the passed formula.
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
  - Get the infeasible subsets the given backend provides.
- const `Model` & `backendsModel` () const
  - Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void `getBackendsModel` () const
- void `getBackendsAllModels` () const
  - Stores all models of a backend in the list of all models of this module.
- virtual `ModuleInput`::iterator `eraseSubformulaFromPassedFormula` (`ModuleInput`::iterator \_subformula, bool \_ignoreOrigins=false)
  - Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void `clearPassedFormula` ()
- std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const
  - Merges the two vectors of sets into the first one.
- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename `Poly`::`PolyType` &\_branchingPolynomial, const `Rational` &\_branchingValue) const
  - Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &&\_premise=std::vector< `FormulaT` >())
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
  - Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- void `splitUnequalConstraint` (const `FormulaT` &)
  - Adds the following lemmas for the given constraint  $p \neq 0$ .
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
  - Adds a formula to the InformationRelevantFormula.
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
  - Gets all InformationRelevantFormulas.
- bool `isLemmaLevel` (`LemmaLevel` level)
  - Checks if current lemma level is greater or equal to given level.

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
  - Checks whether there is no variable assigned by both models.

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`
  - Stores the infeasible subsets.
- `Manager` \*const `mpManager`
  - A reference to the manager.
- `Model` `mModel`

- std::vector< Model > mAllModels
 

*Stores the assignment of the current satisfiable result, if existent.*
- bool mModelComputed
 

*Stores all satisfying assignments.*
- bool mFinalCheck
 

*True, if the model has already been computed.*
- bool mFullCheck
 

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool mObjectiveVariable
 

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable mObjectiveVariable
 

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState
 

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* mBackendsFoundAnswer
 

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals mFoundAnswer
 

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< Module \* > mUsedBackends
 

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< Module \* > mAllBackends
 

*The backends of this module which have been used.*
- std::vector< Lemma > mLemmas
 

*Stores the lemmas being valid formulas this module or its backends made.*
- ModuleInput::iterator mFirstSubformulaToPass
 

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< FormulaT > mConstraintsToInform
 

*Stores the constraints which the backends must be informed about.*
- carl::FastSet< FormulaT > mInformedConstraints
 

*Stores the position of the first constraint of which no backend has been informed about.*
- ModuleInput::const\_iterator mFirstUncheckedReceivedSubformula
 

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned mSmallerMusesCheckCounter
 

*Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > mVariableCounters
 

*Maps variables to the number of their occurrences.*

### 0.15.229.1 Member Typedef Documentation

```
0.15.229.1.1 SettingsType template<typename Settings >
typedef Settings smtrat::GBPPModule< Settings >::SettingsType
```

### 0.15.229.2 Member Enumeration Documentation

```
0.15.229.2.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

**Enumerator**

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

**0.15.229.3 Constructor & Destructor Documentation****0.15.229.3.1 GBPPModule()** `template<typename Settings >`

```
smrat::GBPPModule< Settings >::GBPPModule (
 const ModuleInput * _formula,
 Conditionals & _conditionals,
 Manager * _manager = nullptr)
```

**0.15.229.3.2 ~GBPPModule()** `template<typename Settings >`

```
smrat::GBPPModule< Settings >::~GBPPModule ()
```

**0.15.229.4 Member Function Documentation****0.15.229.4.1 add()** `bool smrat::PModule::add (`

```
 ModuleInput::const_iterator _subformula) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.229.4.2 addConstraintToInform()** `void smrat::Module::addConstraintToInform (`

```
 const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in `smrat::SATModule< Settings >`.

**0.15.229.4.3 addCore()** `virtual bool smrat::Module::addCore (`

```
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.229.4.4 addInformationRelevantFormula()** void smrat::Module::addInformationRelevantFormula (

const [FormulaT](#) & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.229.4.5 addLemma()** void smrat::Module::addLemma (

const [FormulaT](#) & \_lemma,

const [LemmaType](#) & \_lt = [LemmaType::NORMAL](#),

const [FormulaT](#) & \_preferredFormula = [FormulaT\( carl::FormulaType::TRUE \)](#) ) [inline],

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.229.4.6 addOrigin()** void smrat::Module::addOrigin (

[ModuleInput::iterator](#) \_formula,

const [FormulaT](#) & \_origin ) [inline], [protected], [inherited]

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

**0.15.229.4.7 addReceivedSubformulaToPassedFormula()** std::pair<[ModuleInput::iterator](#), bool> smrat::Module::addReceivedSubformulaToPassedFormula (

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

```
0.15.229.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat<→
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.229.4.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat<→
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.229.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.229.4.11 `allModels()`** `const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]`

**Returns**

All satisfying assignments, if existent.

**0.15.229.4.12 `anAnswerFound()`** `bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.229.4.13 `answerFound()`** `const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]`

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.229.4.14 `appliedPreprocessing()`** `bool smtrat::PModule::appliedPreprocessing () const [inline], [inherited]`

**Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.229.4.15 `backendsModel()`** `const Model & smtrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.229.4.16 branchAt() [1/4]** bool smrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.229.4.17 branchAt() [2/4]** bool smrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.229.4.18 branchAt() [3/4]** bool smrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.229.4.19 branchAt() [4/4]** bool smrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.229.4.20 check() Answer smtrat::PModule::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.229.4.21 checkCore() template<typename Settings >
Answer smtrat::GBPPModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.229.4.22 checkInfSubsetForMinimality() void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.229.4.23 checkModel() unsigned smtrat::Module::checkModel () const [protected], [inherited]
```

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.229.4.24 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.229.4.25 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.229.4.26 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.229.4.27 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.229.4.28 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.229.4.29 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.229.4.30 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.229.4.31 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.229.4.32 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.229.4.33 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.229.4.34 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.229.4.35 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.229.4.36 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.229.4.37 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.229.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.229.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.229.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.229.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const` [inline], [inherited]

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.229.4.42 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const` [inline], [inherited]

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.229.4.43 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.229.4.44 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.229.4.45 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.229.4.46 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.229.4.47 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.229.4.48 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.229.4.49 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]

Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.229.4.50 getModelEqualities()** std::list< std::vector< `carl::Variable` > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.229.4.51 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.229.4.52 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.229.4.53 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

**Parameters**

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.229.4.54 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.229.4.55 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.229.4.56 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.229.4.57 `id()`** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.229.4.58 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.229.4.59 `inform()`** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.229.4.60 `informBackends()`** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.229.4.61 `informCore()`** `virtual bool smtrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.229.4.62 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.229.4.63 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.229.4.64 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.229.4.65 isLemmaLevel()** `bool smrat::Module::isLemmaLevel( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.229.4.66 isPreprocessor()** `bool smrat::PModule::isPreprocessor( ) const [inline], [inherited]`

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.229.4.67 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.229.4.68 merge()** `std::vector< FormulaT > smrat::Module::merge( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.229.4.69 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.229.4.70 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (`

```
 const Model & _modelA,
 const Model & _modelB) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.229.4.71 `moduleName()`** `virtual std::string smtrat::Module::moduleName () const [inline], [virtual], [inherited]`**Returns**

The name of the given module type as name.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SplitSOSModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LVEModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::GBModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQPreprocessingModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::BEModule< Settings >](#).

**0.15.229.4.72 `objective()`** `carl::Variable smtrat::Module::objective () const [inline], [inherited]`**0.15.229.4.73 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`

```
 const FormulaT & _origin) const [protected], [inherited]
```

**0.15.229.4.74 passedFormulaBegin()** `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )` [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.229.4.75 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )` [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.229.4.76 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const` [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.229.4.77 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const` [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.229.4.78 print()** `void smtrat::Module::print ( const std::string & _initiation = "***" ) const` [inherited]

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.229.4.79 printAllModels()** `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.229.4.80 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.229.4.81 `printModel()`** `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.229.4.82 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.229.4.83 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.229.4.84 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

**Parameters**

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

**Returns**

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.229.4.85 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.229.4.86 `receivedFormulasAsInfeasibleSubset()`** void smtrat::Module::receivedFormulasAsInfeasibleSubset (

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

**0.15.229.4.87 `receivedVariable()`** bool smtrat::Module::receivedVariable (

```
 carl::Variable::Arg _var) const [inline], [inherited]
```

**0.15.229.4.88 `remove()`** void smtrat::PModule::remove (

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.229.4.89 `removeCore()`** virtual void smtrat::Module::removeCore (

```
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICEModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::GBModule< Settings >](#).

**0.15.229.4.90 `removeOrigin()`** std::pair<ModuleInput::iterator, bool> smtrat::Module::remove←

Origin (

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.229.4.91 `removeOrigins()`** std::pair<ModuleInput::iterator, bool> smtrat::Module::remove←

Origins (

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

#### 0.15.229.4.92 **rPassedFormula()** const [ModuleInput](#)& smtrat::Module::rPassedFormula () const [inline], [inherited]

##### Returns

A reference to the formula passed to the backends of this module.

#### 0.15.229.4.93 **rReceivedFormula()** const [ModuleInput](#)& smtrat::Module::rReceivedFormula () const [inline], [inherited]

##### Returns

A reference to the formula passed to this module.

#### 0.15.229.4.94 **runBackends()** [1/2] virtual [Answer](#) smtrat::PModule::runBackends () [inline], [virtual], [inherited]

Reimplemented from [smtrat::Module](#).

#### 0.15.229.4.95 **runBackends()** [2/2] [Answer](#) smtrat::PModule::runBackends ( bool \_final, bool \_full, carl::Variable \_objective ) [virtual], [inherited]

Runs the backend solvers on the passed formula.

##### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

##### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

#### 0.15.229.4.96 **setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]

Sets this modules unique ID to identify itself.

##### Parameters

|                             |                                    |
|-----------------------------|------------------------------------|
| $\leftrightarrow$<br>$\_id$ | The id to set this module's id to. |
|-----------------------------|------------------------------------|

**0.15.229.4.97 `setThreadPriority()`** `void smtrat::Module::setThreadPriority (`  
`thread_priority _threadPriority )` [inline], [inherited]  
 Sets the priority of this module to get a thread for running its check procedure.

Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.229.4.98 `solverState()`** `Answer smtrat::Module::solverState ( ) const` [inline], [inherited]

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.229.4.99 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint (`  
`const FormulaT & _unequalConstraint )` [protected], [inherited]

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.229.4.100 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const` [inline], [inherited]

Returns

The priority of this module to get a thread for running its check procedure.

**0.15.229.4.101 `updateAllModels()`** `void smtrat::Module::updateAllModels ( )` [virtual], [inherited]

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.229.4.102 `updateLemmas()`** `void smtrat::Module::updateLemmas ( )` [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.229.4.103 `updateModel()`** `template<typename Settings >`

`void smtrat::GBPPModule< Settings >::updateModel ( ) const` [virtual]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.229.4.104 usedBackends()** const std::vector<[Module](#)\*>& smrat::Module::usedBackends ( )  
const [inline], [inherited]

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.229.5 Field Documentation

**0.15.229.5.1 mAllBackends** std::vector<[Module](#)\*> smrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

**0.15.229.5.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.229.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer [protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.229.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.229.5.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.229.5.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.229.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.229.5.8 mFirstUncheckedReceivedSubformula** [ModuleInput](#)::const\_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.229.5.9 mFoundAnswer** [Conditionals](#) smrat::Module::mFoundAnswer [protected], [inherited]

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.229.5.10 mFullCheck** `bool smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.229.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.229.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.229.5.13 mLastBranches** `std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.229.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.229.5.15 mModel** `Model smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.229.5.16 mModelComputed** `bool smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.229.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.229.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.229.5.19 mOldSplittingVariables** `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.229.5.20 mpManager** `Manager* const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.229.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter` [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.229.5.22 mSolverState** std::atomic<[Answer](#)> smrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.229.5.23 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smrat::Module::mTheory← Propagations [protected], [inherited]

**0.15.229.5.24 mUsedBackends** std::vector<[Module\\*](#)> smrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.229.5.25 mVariableCounters** std::vector<std::size\_t> smrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.230 Minisat::Hash< K > Struct Template Reference

```
#include <Map.h>
```

### Public Member Functions

- `uint32_t operator()(const K &k) const`

#### 0.15.230.1 Member Function Documentation

**0.15.230.1.1 operator()** template<class K >  
uint32\_t [Minisat::Hash](#)< K >::operator() (  
 const K & k ) const [inline]

## 0.15.231 smrat::EQModule< Settings >::not\_asserted\_equality::hash Struct Reference

```
#include <EQModule.h>
```

### Public Member Functions

- `std::size_t operator()(const not_asserted_equality &nae) const noexcept`

#### 0.15.231.1 Member Function Documentation

**0.15.231.1.1 operator()** template<typename Settings >  
std::size\_t smrat::EQModule< [Settings](#) >::not\_asserted\_equality::hash::operator() (  
 const not\_asserted\_equality & nae ) const [inline], [noexcept]

## 0.15.232 std::hash< const smrat::expression::ExpressionContent \* > Struct Reference

```
#include <ExpressionHash.h>
```

**Public Member Functions**

- std::size\_t **operator()** (const smtrat::expression::ExpressionContent \*ec) const

**0.15.232.1 Member Function Documentation**

**0.15.232.1.1 operator()()** std::size\_t std::hash< const smtrat::expression::ExpressionContent \* >::operator() ( const smtrat::expression::ExpressionContent \* ec ) const [inline]

**0.15.233 std::hash< smtrat::expression::BinaryExpression > Struct Reference**

```
#include <ExpressionHash.h>
```

**Public Member Functions**

- std::size\_t **operator()** (const smtrat::expression::BinaryExpression &ec) const

**0.15.233.1 Member Function Documentation**

**0.15.233.1.1 operator()()** std::size\_t std::hash< smtrat::expression::BinaryExpression >::operator() ( const smtrat::expression::BinaryExpression & ec ) const [inline]

**0.15.234 std::hash< smtrat::expression::Expression > Struct Reference**

```
#include <ExpressionHash.h>
```

**Public Member Functions**

- std::size\_t **operator()** (const smtrat::expression::Expression &expr) const

**0.15.234.1 Member Function Documentation**

**0.15.234.1.1 operator()()** std::size\_t std::hash< smtrat::expression::Expression >::operator() ( const smtrat::expression::Expression & expr ) const [inline]

**0.15.235 std::hash< smtrat::expression::ITEExpression > Struct Reference**

```
#include <ExpressionHash.h>
```

**Public Member Functions**

- std::size\_t **operator()** (const smtrat::expression::ITEExpression &ec) const

**0.15.235.1 Member Function Documentation**

**0.15.235.1.1 operator()()** std::size\_t std::hash< smtrat::expression::ITEExpression >::operator() ( const smtrat::expression::ITEExpression & ec ) const [inline]

## 0.15.236 std::hash< smtrat::expression::NaryExpression > Struct Reference

```
#include <ExpressionHash.h>
```

### Public Member Functions

- std::size\_t `operator()` (const `smtrat::expression::NaryExpression` &ec) const

#### 0.15.236.1 Member Function Documentation

```
0.15.236.1.1 operator()() std::size_t std::hash< smtrat::expression::NaryExpression >::operator()
(
 const smtrat::expression::NaryExpression & ec) const [inline]
```

## 0.15.237 std::hash< smtrat::expression::QuantifierExpression > Struct Reference

```
#include <ExpressionHash.h>
```

### Public Member Functions

- std::size\_t `operator()` (const `smtrat::expression::QuantifierExpression` &ec) const

#### 0.15.237.1 Member Function Documentation

```
0.15.237.1.1 operator()() std::size_t std::hash< smtrat::expression::QuantifierExpression >::operator()
(
 const smtrat::expression::QuantifierExpression & ec) const [inline]
```

## 0.15.238 std::hash< smtrat::expression::UnaryExpression > Struct Reference

```
#include <ExpressionHash.h>
```

### Public Member Functions

- std::size\_t `operator()` (const `smtrat::expression::UnaryExpression` &ec) const

#### 0.15.238.1 Member Function Documentation

```
0.15.238.1.1 operator()() std::size_t std::hash< smtrat::expression::UnaryExpression >::operator()
(
 const smtrat::expression::UnaryExpression & ec) const [inline]
```

## 0.15.239 std::hash< smtrat::Ira::Variable< T1, T2 > > Struct Template Reference

Implements std::hash for sort value.

```
#include <Variable.h>
```

### Public Member Functions

- size\_t `operator()` (const `smtrat::Ira::Variable< T1, T2 >` &\_IraVar) const

### 0.15.239.1 Detailed Description

```
template<typename T1, typename T2>
struct std::hash< smrat::lra::Variable< T1, T2 > >
```

Implements std::hash for sort value.

### 0.15.239.2 Member Function Documentation

```
0.15.239.2.1 operator() template<typename T1 , typename T2 >
size_t std::hash< smrat::lra::Variable< T1, T2 > >::operator() (
 const smrat::lra::Variable< T1, T2 > & _lraVar) const [inline]
```

#### Parameters

|                      |                                       |
|----------------------|---------------------------------------|
| <code>_lraVar</code> | The LRA-variable to get the hash for. |
|----------------------|---------------------------------------|

#### Returns

The hash of the given LRA-variable.

## 0.15.240 std::hash< smrat::vs::Substitution > Struct Reference

```
#include <Substitution.h>
```

#### Public Member Functions

- `size_t operator()(const smrat::vs::Substitution &_substitution) const`

### 0.15.240.1 Member Function Documentation

```
0.15.240.1.1 operator() size_t std::hash< smrat::vs::Substitution >::operator() (
 const smrat::vs::Substitution & _substitution) const [inline]
```

## 0.15.241 std::hash< std::vector< T > > Struct Template Reference

```
#include <ExpressionHash.h>
```

#### Public Member Functions

- `std::size_t operator()(const std::vector< T > &v) const`

### 0.15.241.1 Member Function Documentation

```
0.15.241.1.1 operator() template<typename T >
std::size_t std::hash< std::vector< T > >::operator() (
 const std::vector< T > & v) const [inline]
```

## 0.15.242 std::hash\_combiner Struct Reference

```
#include <ExpressionHash.h>
```

## Public Member Functions

- `hash_combiner (std::size_t _seed=0)`
- `hash_combiner & operator() (std::size_t h)`
- template<typename T >  
  `hash_combiner & operator() (const T &t)`
- `operator std::size_t () const`

## Data Fields

- `std::size_t seed`

### 0.15.242.1 Constructor & Destructor Documentation

**0.15.242.1.1 `hash_combiner()`** `std::hash_combiner::hash_combiner ( std::size_t _seed = 0 ) [inline]`

### 0.15.242.2 Member Function Documentation

**0.15.242.2.1 `operator std::size_t()`** `std::hash_combiner::operator std::size_t () const [inline]`

**0.15.242.2.2 `operator()()` [1/2]** `template<typename T > hash_combiner& std::hash_combiner::operator() ( const T &t ) [inline]`

**0.15.242.2.3 `operator()()` [2/2]** `hash_combiner& std::hash_combiner::operator() ( std::size_t h ) [inline]`

### 0.15.242.3 Field Documentation

**0.15.242.3.1 `seed`** `std::size_t std::hash_combiner::seed`

## 0.15.243 Minisat::Heap< Comp > Class Template Reference

```
#include <Heap.h>
```

## Public Member Functions

- `Heap (const Comp &c)`
- `int size () const`
- `bool empty () const`
- `bool inHeap (int n) const`
- `int operator[] (int index) const`
- `void decrease (int n)`
- `void increase (int n)`
- `void update (int n)`
- `void insert (int n)`
- `int removeMin ()`
- `void build (vec< int > &ns)`
- `void clear (bool dealloc=false)`
- `void print () const`

**0.15.243.1 Constructor & Destructor Documentation**

**0.15.243.1.1 `Heap()`** template<class Comp >  
Minisat::Heap< Comp >::Heap ( const Comp & c ) [inline]

**0.15.243.2 Member Function Documentation**

**0.15.243.2.1 `build()`** template<class Comp >  
void Minisat::Heap< Comp >::build ( vec< int > & ns ) [inline]

**0.15.243.2.2 `clear()`** template<class Comp >  
void Minisat::Heap< Comp >::clear ( bool deallocate = false ) [inline]

**0.15.243.2.3 `decrease()`** template<class Comp >  
void Minisat::Heap< Comp >::decrease ( int n ) [inline]

**0.15.243.2.4 `empty()`** template<class Comp >  
bool Minisat::Heap< Comp >::empty ( ) const [inline]

**0.15.243.2.5 `increase()`** template<class Comp >  
void Minisat::Heap< Comp >::increase ( int n ) [inline]

**0.15.243.2.6 `inHeap()`** template<class Comp >  
bool Minisat::Heap< Comp >::inHeap ( int n ) const [inline]

**0.15.243.2.7 `insert()`** template<class Comp >  
void Minisat::Heap< Comp >::insert ( int n ) [inline]

**0.15.243.2.8 `operator[]()`** template<class Comp >  
int Minisat::Heap< Comp >::operator[] ( int index ) const [inline]

**0.15.243.2.9 `print()`** template<class Comp >  
void Minisat::Heap< Comp >::print ( ) const [inline]

**0.15.243.2.10 removeMin()** template<class Comp >  
int Minisat::Heap< Comp >::removeMin ( ) [inline]

**0.15.243.2.11 size()** template<class Comp >  
int Minisat::Heap< Comp >::size ( ) const [inline]

**0.15.243.2.12 update()** template<class Comp >  
void Minisat::Heap< Comp >::update ( int n ) [inline]

## 0.15.244 smtrat::parser::HexadecimalParser Struct Reference

Parses hexadecimals: #x [0-9a-fA-F] +  
#include <Lexicon.h>

### Public Types

- `typedef boost::iterator_range< Iterator > ITRange`

### Public Member Functions

- `HexadecimalParser ()`
- `FixedWidthConstant< Integer > build (const ITRange &itr, const Integer &val)`

### Data Fields

- `qi::uint_parser< Integer, 16, 1,-1 > number`
- `qi::rule< Iterator, FixedWidthConstant< Integer >, Skipper, qi::locals< Integer > > main`
- `qi::rule< Iterator, FixedWidthConstant< Integer >, Skipper > main2`

### 0.15.244.1 Detailed Description

Parses hexadecimals: #x [0-9a-fA-F] +

### 0.15.244.2 Member Typedef Documentation

**0.15.244.2.1 ITRange** `typedef boost::iterator_range<Iterator> smtrat::parser::HexadecimalParser::ITRange`

### 0.15.244.3 Constructor & Destructor Documentation

**0.15.244.3.1 HexadecimalParser()** `smtrat::parser::HexadecimalParser::HexadecimalParser ( ) [inline]`

### 0.15.244.4 Member Function Documentation

**0.15.244.4.1 build()** `FixedWidthConstant<Integer> smtrat::parser::HexadecimalParser::build ( const ITRange & itr, const Integer & val ) [inline]`

### 0.15.244.5 Field Documentation

**0.15.244.5.1 main** `qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper, qi::locals<Integer>> smrat::parser::HexadecimalParser::main`

**0.15.244.5.2 main2** `qi::rule<Iterator, FixedWidthConstant<Integer>), Skipper> smrat::parser<::HexadecimalParser::main2`

**0.15.244.5.3 number** `qi::uint_parser<Integer,16,1,-1> smrat::parser::HexadecimalParser<::number`

## 0.15.245 smrat::icp::HistoryNode Class Reference

```
#include <HistoryNode.h>
```

### Public Member Functions

- `HistoryNode ()`

*Methods.*
- `HistoryNode (const EvalDoubleIntervalMap &_intervals)`
- `~HistoryNode ()`
- `const EvalDoubleIntervalMap & intervals () const`

*Getters and Setters.*
- `EvalDoubleIntervalMap & rIntervals ()`
- `DoubleInterval & getInterval (carl::Variable::Arg _variable)`
- `void setIntervals (const EvalDoubleIntervalMap &_map)`
- `bool addInterval (carl::Variable::Arg _var, const DoubleInterval &_interval)`

*updates or inserts an interval into the actual map*
- `bool hasEmptyInterval () const`
- `std::set< const ContractionCandidate * > appliedContractions ()`
- `FormulaSetT appliedConstraints ()`
- `void appliedConstraints (std::vector< FormulaT > &_result)`
- `std::set< ConstraintT > & rStateInfeasibleConstraints ()`
- `const std::set< ConstraintT > & stateInfeasibleConstraints () const`
- `set_icpVariable & rStateInfeasibleVariables ()`
- `set_icpVariable stateInfeasibleVariables () const`
- `bool addInfeasibleConstraint (const ConstraintT &_constraint, bool _addOnlyConstraint=false)`
- `bool addInfeasibleVariable (const IcpVariable *_variable, bool _addOnlyVariable=false)`
- `void addContraction (ContractionCandidate *_candidate, const set_icpVariable &_variables)`
- `const std::set< const ContractionCandidate * > & getCandidates () const`
- `std::map< carl::Variable, std::set< ConstraintT > > & rReasons ()`
- `const std::map< carl::Variable, std::set< ConstraintT > > & reasons () const`
- `std::set< ConstraintT > & reasons (carl::Variable::Arg _variable)`
- `void addReason (carl::Variable::Arg _variable, const ConstraintT &_reason)`
- `void addReasons (carl::Variable::Arg _variable, const std::set< ConstraintT > &_reasons)`
- `void addReasons (carl::Variable::Arg _variable, const FormulasT &_origins)`
- `void addVariableReason (carl::Variable::Arg _variable, const IcpVariable *_reason)`
- `void addVariableReasons (carl::Variable::Arg _variable, set_icpVariable _variables)`
- `const std::map< carl::Variable, set_icpVariable > & variableReasons ()`
- `std::map< carl::Variable, set_icpVariable > & rVariableReasons ()`
- `set_icpVariable variableReasons (carl::Variable::Arg _variable)`

- void `variableHull` (carl::Variable::Arg `_variable`, `set_icpVariable` & `_result`) const
- void `propagateStateInfeasibleConstraints` (`HistoryNode` \* `_target`) const
- void `propagateStateInfeasibleVariables` (`HistoryNode` \* `_target`) const
- void `print` (std::ostream & `_out`=std::cout) const
- void `printReasons` (std::ostream & `_out`=std::cout) const
- void `printVariableReasons` (std::ostream & `_out`=std::cout) const
- void `reset` ()

### 0.15.245.1 Constructor & Destructor Documentation

**0.15.245.1.1 HistoryNode()** [1/2] smtrat::icp::HistoryNode::HistoryNode ( ) [inline]  
Methods.

**0.15.245.1.2 HistoryNode()** [2/2] smtrat::icp::HistoryNode::HistoryNode ( const `EvalDoubleIntervalMap` & `_intervals` ) [inline]

**0.15.245.1.3 ~HistoryNode()** smtrat::icp::HistoryNode::~HistoryNode ( ) [inline]

### 0.15.245.2 Member Function Documentation

**0.15.245.2.1 addContraction()** void smtrat::icp::HistoryNode::addContraction ( `ContractionCandidate` \* `_candidate`, const `set_icpVariable` & `_variables` ) [inline]

**0.15.245.2.2 addInfeasibleConstraint()** bool smtrat::icp::HistoryNode::addInfeasibleConstraint ( const `ConstraintT` & `_constraint`, bool `_addOnlyConstraint` = false ) [inline]

**0.15.245.2.3 addInfeasibleVariable()** bool smtrat::icp::HistoryNode::addInfeasibleVariable ( const `IcpVariable` \* `_variable`, bool `_addOnlyVariable` = false ) [inline]

**0.15.245.2.4 addInterval()** bool smtrat::icp::HistoryNode::addInterval ( carl::Variable::Arg `_var`, const `DoubleInterval` & `_interval` ) [inline]  
updates or inserts an interval into the actual map

#### Parameters

|                        |  |
|------------------------|--|
| <code>_var</code>      |  |
| <code>_interval</code> |  |

#### Returns

true if only an update

**0.15.245.2.5 `addReason()`** void smtrat::icp::HistoryNode::addReason ( carl::Variable::Arg \_variable, const ConstraintT & \_reason ) [inline]

**0.15.245.2.6 `addReasons()` [1/2]** void smtrat::icp::HistoryNode::addReasons ( carl::Variable::Arg \_variable, const FormulasT & \_origins ) [inline]

**0.15.245.2.7 `addReasons()` [2/2]** void smtrat::icp::HistoryNode::addReasons ( carl::Variable::Arg \_variable, const std::set< ConstraintT > & \_reasons ) [inline]

**0.15.245.2.8 `addVariableReason()`** void smtrat::icp::HistoryNode::addVariableReason ( carl::Variable::Arg \_variable, const IcpVariable \* \_reason ) [inline]

**0.15.245.2.9 `addVariableReasons()`** void smtrat::icp::HistoryNode::addVariableReasons ( carl::Variable::Arg \_variable, set\_icpVariable \_variables ) [inline]

**0.15.245.2.10 `appliedConstraints()` [1/2]** FormulaSetT smtrat::icp::HistoryNode::appliedConstraints ( ) [inline]

**0.15.245.2.11 `appliedConstraints()` [2/2]** void smtrat::icp::HistoryNode::appliedConstraints ( std::vector< FormulaT > & \_result ) [inline]

**0.15.245.2.12 `appliedContractions()`** std::set<const ContractionCandidate\*> smtrat::icp::HistoryNode::appliedContractions ( ) [inline]

**0.15.245.2.13 `getCandidates()`** const std::set<const ContractionCandidate\*>& smtrat::icp::HistoryNode::getCandidates ( ) const [inline]

**0.15.245.2.14 `getInterval()`** DoubleInterval& smtrat::icp::HistoryNode::getInterval ( carl::Variable::Arg \_variable ) [inline]

**0.15.245.2.15 `hasEmptyInterval()`** bool smtrat::icp::HistoryNode::hasEmptyInterval ( ) const [inline]

**0.15.245.2.16 `intervals()`** const EvalDoubleIntervalMap& smtrat::icp::HistoryNode::intervals ( ) const [inline]  
Getters and Setters.

**0.15.245.2.17 `print()`** void smtrat::icp::HistoryNode::print ( std::ostream & \_out = std::cout ) const [inline]

**0.15.245.2.18 `printReasons()`** void smtrat::icp::HistoryNode::printReasons ( std::ostream & \_out = std::cout ) const [inline]

**0.15.245.2.19 `printVariableReasons()`** void smtrat::icp::HistoryNode::printVariableReasons ( std::ostream & \_out = std::cout ) const [inline]

**0.15.245.2.20 `propagateStateInfeasibleConstraints()`** void smtrat::icp::HistoryNode::propagateStateInfeasibleConstraints ( HistoryNode \* \_target ) const [inline]

**0.15.245.2.21 `propagateStateInfeasibleVariables()`** void smtrat::icp::HistoryNode::propagateStateInfeasibleVariables ( HistoryNode \* \_target ) const [inline]

**0.15.245.2.22 `reasons() [1/2]`** const std::map<carl::Variable, std::set<ConstraintT> >& smtrat::icp::HistoryNode::reasons ( ) const [inline]

**0.15.245.2.23 `reasons() [2/2]`** std::set<ConstraintT>& smtrat::icp::HistoryNode::reasons ( carl::Variable::Arg \_variable ) [inline]

**0.15.245.2.24 `reset()`** void smtrat::icp::HistoryNode::reset ( ) [inline]

**0.15.245.2.25 `rIntervals()`** EvalDoubleIntervalMap& smtrat::icp::HistoryNode::rIntervals ( ) [inline]

**0.15.245.2.26 `rReasons()`** std::map<carl::Variable, std::set<ConstraintT> >& smtrat::icp::HistoryNode::rReasons ( ) [inline]

**0.15.245.2.27 `rStateInfeasibleConstraints()`** std::set<ConstraintT>& smtrat::icp::HistoryNode::rStateInfeasibleConstraints ( ) [inline]

**0.15.245.2.28 `rStateInfeasibleVariables()`** set\_icpVariable& smtrat::icp::HistoryNode::rStateInfeasibleVariables ( ) [inline]

**0.15.245.2.29 `rVariableReasons()`** std::map<carl::Variable, set\_icpVariable>& smtrat::icp::HistoryNode::rVariableReasons ( ) [inline]

**0.15.245.2.30 `setIntervals()`** void smtrat::icp::HistoryNode::setIntervals ( const EvalDoubleIntervalMap & \_map ) [inline]

---

**0.15.245.2.31 stateInfeasibleConstraints()** const std::set<ConstraintT>& smtrat::icp::HistoryNode::stateInfeasibleConstraints ( ) const [inline]

**0.15.245.2.32 stateInfeasibleVariables()** set\_icpVariable smtrat::icp::HistoryNode::stateInfeasibleVariables ( ) const [inline]

**0.15.245.2.33 variableHull()** void smtrat::icp::HistoryNode::variableHull ( carl::Variable::Arg \_variable, set\_icpVariable & \_result ) const [inline]

**0.15.245.2.34 variableReasons() [1/2]** const std::map<carl::Variable, set\_icpVariable>& smtrat::icp::HistoryNode::variableReasons ( ) [inline]

**0.15.245.2.35 variableReasons() [2/2]** set\_icpVariable smtrat::icp::HistoryNode::variableReasons ( carl::Variable::Arg \_variable ) [inline]

## 0.15.246 smtrat::ICEModule< Settings > Class Template Reference

#include <ICEModule.h>

### Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

### Public Member Functions

- ICEModule (const ModuleInput \* \_formula, Conditionals & \_conditionals, Manager \* \_manager=nullptr)
- ~ICEModule ()
- bool addCore (ModuleInput::const\_iterator \_subformula)
 

*The module has to take the given sub-formula of the received formula into account.*
- void removeCore (ModuleInput::const\_iterator \_subformula)
 

*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- Answer checkCore ()
 

*Checks the received formula for consistency.*
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const\_iterator \_subformula)
 

*The module has to take the given sub-formula of the received formula into account.*
- void remove (ModuleInput::const\_iterator \_subformula)
 

*Removes everything related to the given sub-formula of the received formula.*
- Answer check (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)
 

*Checks the received formula for consistency.*
- Answer runBackends (bool \_final, bool \_full, carl::Variable \_objective)
 

*Runs the backend solvers on the passed formula.*
- virtual Answer runBackends ()
- std::pair< bool, FormulaT > getReceivedFormulaSimplified ()
 

*Updates the current assignment into the model.*
- void updateModel () const
 

*Updates the current assignment into the model.*
- bool inform (const FormulaT & \_constraint)

- **Informs the module about the given constraint.**
  - void **deinform** (const **FormulaT** &\_constraint)  
*The inverse of informing about a constraint.*
  - virtual void **init** ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
  - virtual void **updateAllModels** ()  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
  - virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
  - unsigned **currentlySatisfiedByBackend** (const **FormulaT** &\_formula) const
  - virtual unsigned **currentlySatisfied** (const **FormulaT** &) const
  - bool **receivedVariable** (carl::Variable::Arg \_var) const
  - **Answer solverState** () const
  - std::size\_t **id** () const
  - void **setId** (std::size\_t \_id)  
*Sets this modules unique ID to identify itself.*
  - **thread\_priority threadPriority** () const
  - void **setThreadPriority** (**thread\_priority** \_threadPriority)  
*Sets the priority of this module to get a thread for running its check procedure.*
  - const **ModuleInput** \* **pReceivedFormula** () const
  - const **ModuleInput** & **rReceivedFormula** () const
  - const **ModuleInput** \* **pPassedFormula** () const
  - const **ModuleInput** & **rPassedFormula** () const
  - const **Model** & **model** () const
  - const std::vector< **Model** > & **allModels** () const
  - const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
  - const std::vector< **Module** \* > & **usedBackends** () const
  - const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
  - const carl::FastSet< **FormulaT** > & **informedConstraints** () const
  - void **addLemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lt= **LemmaType**::NORMAL, const **FormulaT** &\_preferredFormula= **FormulaT**(carl::FormulaType::TRUE))  
*Stores a lemma being a valid formula.*
  - bool **hasLemmas** ()  
*Checks whether this module or any of its backends provides any lemmas.*
  - void **clearLemmas** ()  
*Deletes all yet found lemmas.*
  - const std::vector< **Lemma** > & **lemmas** () const
  - **ModuleInput**::const\_iterator **firstUncheckedReceivedSubformula** () const
  - **ModuleInput**::const\_iterator **firstSubformulaToPass** () const
  - void **receivedFormulaChecked** ()  
*Notifies that the received formulas has been checked.*
  - const **smrat::Conditionals** & **answerFound** () const
  - virtual std::string **moduleName** () const
  - carl::Variable **objective** () const
  - bool **is\_minimizing** () const
  - void **excludeNotReceivedVariablesFromModel** () const  
*Excludes all variables from the current model, which do not occur in the received formula.*
  - void **updateLemmas** ()  
*Stores all lemmas of any backend of this module in its own lemma vector.*
  - void **collectTheoryPropagations** ()
  - void **collectOrigins** (const **FormulaT** &\_formula, **FormulasT** &\_origins) const  
*Collects the formulas in the given formula, which are part of the received formula.*
  - void **collectOrigins** (const **FormulaT** &\_formula, **FormulaSetT** &\_origins) const

- bool `isValidInfeasibleSubset () const`
- void `checkInfeasibleSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _insubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore = 5`

*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`

*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches = 0`

*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`

*Reusable splitting variables.*

### Protected Member Functions

- virtual bool `informCore (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- virtual void `deinformCore (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- bool `anAnswerFound () const`

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel () const`

*Clears the assignment, if any was found.*
- void `clearModels () const`

*Clears all assignments, if any was found.*
- void `cleanModel () const`

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin ()`
- `ModuleInput::iterator passedFormulaEnd ()`
- void `addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`

- Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`

*Gets the origins of the passed formula at the given position.*
  - void `getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
  - void `getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
  - std::pair< `ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
  - std::pair< `ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
  - void `informBackends (const FormulaT &_constraint)`

*Informs all backends of this module about the given constraint.*
  - virtual void `addConstraintToInform (const FormulaT &_constraint)`

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
  - std::pair< `ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`

*Copies the given sub-formula of the received formula to the passed formula.*
  - bool `originInReceivedFormula (const FormulaT &_origin) const`
  - std::pair< `ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)`

*Adds the given formula to the passed formula with no origin.*
  - std::pair< `ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`

*Adds the given formula to the passed formula.*
  - std::pair< `ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`

*Adds the given formula to the passed formula.*
  - void `generateTrivialInfeasibleSubset ()`

*Stores the trivial infeasible subset being the set of received formulas.*
  - void `receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

*Stores an infeasible subset consisting only of the given received formula.*
  - std::vector< `FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const`
  - void `getInfeasibleSubsets ()`

*Copies the infeasible subsets of the passed formula.*
  - std::vector< `FormulaSetT > getInfeasibleSubsets (const Module &_backend) const`

*Get the infeasible subsets the given backend provides.*
  - const `Model & backendsModel () const`

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - void `getBackendsModel () const`
  - void `getBackendsAllModels () const`

*Stores all models of a backend in the list of all models of this module.*
  - virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
  - void `clearPassedFormula ()`
  - std::vector< `FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`

*Merges the two vectors of sets into the first one.*
  - size\_t `determine_smallest_origin (const std::vector< FormulaT > &origins) const`
  - bool `probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const`

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
  - bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`

- Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
  - bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector<`FormulaT`> &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
  - bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
    - void `splitUnequalConstraint` (const `FormulaT` &
 

Adds the following lemmas for the given constraint  $p \neq 0$ .
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
  - Adds a formula to the InformationRelevantFormula.
- const std::set<`FormulaT`> & `getInformationRelevantFormulas` ()
  - Gets all InformationRelevantFormulas.
- bool `isLemmaLevel` (`LemmaLevel` level)
  - Checks if current lemma level is greater or equal to given level.

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
  - Checks whether there is no variable assigned by both models.

## Protected Attributes

- std::vector<`FormulaSetT`> `mInfeasibleSubsets`
  - Stores the infeasible subsets.
- `Manager` \*const `mpManager`
  - A reference to the manager.
- `Model` `mModel`
  - Stores the assignment of the current satisfiable result, if existent.
- std::vector<`Model`> `mAllModels`
  - Stores all satisfying assignments.
- bool `mModelComputed`
  - True, if the model has already been computed.
- bool `mFinalCheck`
  - true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool `mFullCheck`
  - false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable `mObjectiveVariable`
  - Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.
- std::vector<`TheoryPropagation`> `mTheoryPropagations`
- std::atomic<`Answer`> `mSolverState`
  - States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic\_bool \* `mBackendsFoundAnswer`
  - This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- `Conditionals` `mFoundAnswer`
  - Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector<`Module`\*> `mUsedBackends`
  - The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

- `std::vector< Module * > mAllBackends`  
*The backends of this module which have been used.*
- `std::vector< Lemma > mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- `carl::FastSet< FormulaT > mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.246.1 Member Typedef Documentation

**0.15.246.1.1 SettingsType** `template<typename Settings >`  
`typedef Settings smtrat::ICEModule< Settings >::SettingsType`

### 0.15.246.2 Member Enumeration Documentation

**0.15.246.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.246.3 Constructor & Destructor Documentation

**0.15.246.3.1 ICEModule()** `template<typename Settings >`  
`smtrat::ICEModule< Settings >::ICEModule (`  
`const ModuleInput * _formula,`  
`Conditionals & _conditionals,`  
`Manager * _manager = nullptr )`

**0.15.246.3.2 ~ICEModule()** `template<typename Settings >`  
`smtrat::ICEModule< Settings >::~ICEModule ( )`

### 0.15.246.4 Member Function Documentation

**0.15.246.4.1 add()** `bool smtrat::PModule::add (`  
`ModuleInput::const_iterator _subformula )` [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.246.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`  
`const FormulaT & _constraint )` [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.246.4.3 addCore()** `template<typename Settings >`  
`bool smtrat::ICEModule< Settings >::addCore (`  
`ModuleInput::const_iterator _subformula )` [virtual]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.246.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula (`  
`const FormulaT & formula )` [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.246.4.5 addLemma()** `void smtrat::Module::addLemma (`  
`const FormulaT & _lemma,`  
`const LemmaType & _lt = LemmaType::NORMAL,`

```
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
Stores a lemma being a valid formula.
```

**Parameters**

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| _lemma            | The eduction/lemma to store.                                          |
| _lt               | The type of the lemma.                                                |
| _preferredFormula | Hint for the next decision, which formula should be assigned to true. |

**0.15.246.4.6 addOrigin()** void smtrat::Module::addOrigin (

```
ModuleInput::iterator _formula,
const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

**0.15.246.4.7 addReceivedSubformulaToPassedFormula()** std::pair<ModuleInput::iterator,bool>

```
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

**0.15.246.4.8 addSubformulaToPassedFormula() [1/3]** std::pair<ModuleInput::iterator,bool> smtrat<->

```
::Module::addSubformulaToPassedFormula (
```

```
const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.246.4.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                 |
| <i>_origin</i>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.246.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector<FormulaT>> & _origins) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.246.4.11 allModels() const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]
```

**Returns**

All satisfying assignments, if existent.

```
0.15.246.4.12 anAnswerFound() bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.246.4.13 answerFound()** const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.246.4.14 appliedPreprocessing()** bool smtrat::PModule::appliedPreprocessing ( ) const [inline], [inherited]

**Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.246.4.15 backendsModel()** const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.246.4.16 branchAt() [1/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & \_premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]

**0.15.246.4.17 branchAt() [2/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, std::vector< FormulaT > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.246.4.18 branchAt() [3/4]** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & \_premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]

**0.15.246.4.19 branchAt() [4/4]** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, std::vector< FormulaT > && \_premise,

```

 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]

```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

#### 0.15.246.4.20 `check()` Answer `smtrat::PModule::check (`

```

 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]

```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

#### 0.15.246.4.21 `checkCore()` template<typename Settings >

```
Answer smtrat::ICEModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.246.4.22 checkInfSubsetForMinimality() void smrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infsSubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.246.4.23 checkModel() unsigned smrat::Module::checkModel () const [protected], [inherited]
```

#### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

```
0.15.246.4.24 cleanModel() void smrat::Module::cleanModel () const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

```
0.15.246.4.25 clearLemmas() void smrat::Module::clearLemmas () [inline], [inherited]
```

Deletes all yet found lemmas.

```
0.15.246.4.26 clearModel() void smrat::Module::clearModel () const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

```
0.15.246.4.27 clearModels() void smrat::Module::clearModels () const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

```
0.15.246.4.28 clearPassedFormula() void smrat::Module::clearPassedFormula () [protected], [inherited]
```

```
0.15.246.4.29 collectOrigins() [1/2] void smrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

```
0.15.246.4.30 collectOrigins() [2/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                 |                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <i>_origins</i> | The set in which to store the origins.                                                               |

```
0.15.246.4.31 collectTheoryPropagations() void smtrat::Module::collectTheoryPropagations () [inherited]
```

```
0.15.246.4.32 constraintsToInform() const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]
```

#### Returns

The constraints which the backends must be informed about.

```
0.15.246.4.33 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

```
0.15.246.4.34 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.246.4.35 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                    |                                                         |
|--------------------|---------------------------------------------------------|
| <i>_constraint</i> | The constraint to remove from internal data structures. |
|--------------------|---------------------------------------------------------|

**0.15.246.4.36 `deinformCore()`** `virtual void smtrat::Module::deinformCore (`  
    `const FormulaT & _constraint )` [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.246.4.37 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (`  
    `const std::vector< FormulaT > & origins ) const` [protected], [inherited]

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.246.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`  
    `ModuleInput::iterator _subformula,`  
    `bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.246.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.246.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`  
    `const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

**0.15.246.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.246.4.42 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.246.4.43 `freeSplittingVariable()`** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable) [inline], [static], [inherited]`

**0.15.246.4.44 `generateTrivialInfeasibleSubset()`** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.246.4.45 `getBackendsAllModels()`** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.246.4.46 `getBackendsModel()`** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.246.4.47 `getInfeasibleSubsets() [1/2]`** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.246.4.48 `getInfeasibleSubsets() [2/2]`** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.246.4.49 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.246.4.50 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.246.4.51 `getOrigins() [1/3]`** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.246.4.52 `getOrigins() [2/3]`** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.246.4.53 `getOrigins() [3/3]`** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.246.4.54 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified( ) [virtual], [inherited]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.246.4.55 `hasLemmas()`** `bool smtrat::Module::hasLemmas( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

**0.15.246.4.56 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.246.4.57 `id()`** `std::size_t smtrat::Module::id( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.246.4.58 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.246.4.59 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.246.4.60 informBackends()** void smrat::Module::informBackends ( const FormulaT & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.246.4.61 informCore()** virtual bool smrat::Module::informCore (

const FormulaT & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.246.4.62 informedConstraints()** const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.246.4.63 init()** void smrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

**0.15.246.4.64 is\_minimizing()** bool smrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.246.4.65 `isLemmaLevel()`** `bool smtrat::Module::isLemmaLevel (``LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.246.4.66 `isPreprocessor()`** `bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.246.4.67 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.246.4.68 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (``const std::vector< FormulaT > & _vecSetA,``const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.246.4.69 `model()`** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.246.4.70 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (``const Model & _modelA,``const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.246.4.71 `moduleName()`** `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

**Returns**

The name of the given module type as name.

Reimplemented in `smtrat::VSModule< Settings >`, `smtrat::UnionFindModule< Settings >`, `smtrat::UFCegarModule< Settings >`, `smtrat::STropModule< Settings >`, `smtrat::SplitSOSModule< Settings >`, `smtrat::PBPPModule< Settings >`, `smtrat::PBGaussModule< Settings >`, `smtrat::NRAILModule< Settings >`, `smtrat::NewCADModule< Settings >`, `smtrat::LVEModule< Settings >`, `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, `smtrat::LRAModule< LRASettings1 >`, `smtrat::IntEqModule< Settings >`, `smtrat::IntBlastModule< Settings >`, `smtrat::IncWidthModule< Settings >`, `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, `smtrat::ICPModule< ICPSettings1 >`, `smtrat::GBModule< Settings >`, `smtrat::FouMoModule< Settings >`, `smtrat::EQPreprocessingModule< Settings >`, `smtrat::EQModule< Settings >`, `smtrat::CurryModule< Settings >`, `smtrat::CubeLIAModule< Settings >`, `smtrat::CSplitModule< Settings >`, `smtrat::BVMModule< Settings >`, and `smtrat::BEModule< Settings >`.

**0.15.246.4.72 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.246.4.73 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.246.4.74 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.246.4.75 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.246.4.76 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.246.4.77 pReceivedFormula()** const `ModuleInput*` smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.246.4.78 print()** void smtrat::Module::print ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.246.4.79 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & `_out` = `std::cout` ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.246.4.80 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.246.4.81 printModel()** void smtrat::Module::printModel ( std::ostream & `_out` = `std::cout` ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.246.4.82 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.246.4.83 printReceivedFormula()** void smtrat::Module::printReceivedFormula (const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

## Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.246.4.84 probablyLooping()** bool smtrat::Module::probablyLooping (const typename Poly::PolyType & *\_branchingPolynomial*,const Rational & *\_branchingValue* ) const [protected], [inherited]

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

## Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

## Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.246.4.85 receivedFormulaChecked()** void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.246.4.86 receivedFormulasAsInfeasibleSubset()** void smtrat::Module::receivedFormulasAsInfeasibleSubset (ModuleInput::const\_iterator *\_subformula* ) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

**0.15.246.4.87 receivedVariable()** bool smtrat::Module::receivedVariable (carl::Variable::Arg *\_var* ) const [inline], [inherited]**0.15.246.4.88 remove()** void smtrat::PModule::remove (ModuleInput::const\_iterator *\_subformula* ) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

## Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub formula of the received formula to remove. |
|--------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

```
0.15.246.4.89 removeCore() template<typename Settings >
void smtrat::ICEModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.  
Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

```
0.15.246.4.90 removeOrigin() std::pair<ModuleInput::iterator,bool> smtrat::Module::remove<-
Origin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

```
0.15.246.4.91 removeOrigins() std::pair<ModuleInput::iterator,bool> smtrat::Module::remove<-
Origins (
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

```
0.15.246.4.92 rPassedFormula() const ModuleInput& smtrat::Module::rPassedFormula () const
[inline], [inherited]
```

#### Returns

A reference to the formula passed to the backends of this module.

```
0.15.246.4.93 rReceivedFormula() const ModuleInput& smtrat::Module::rReceivedFormula () const
[inline], [inherited]
```

#### Returns

A reference to the formula passed to this module.

```
0.15.246.4.94 runBackends() [1/2] virtual Answer smtrat::PModule::runBackends () [inline],
[virtual], [inherited]
```

Reimplemented from [smtrat::Module](#).

```
0.15.246.4.95 runBackends() [2/2] Answer smtrat::PModule::runBackends (
 bool _final,
 bool _full,
 carl::Variable _objective) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                     |                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------|
| <code>_final</code> | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT. |
|---------------------|---------------------------------------------------------------------------------------------------------|

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.246.4.96 `setId()`** `void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

**Parameters**

|                                                      |                                    |
|------------------------------------------------------|------------------------------------|
| <code>←</code><br><code>—←</code><br><code>id</code> | The id to set this module's id to. |
|------------------------------------------------------|------------------------------------|

**0.15.246.4.97 `setThreadPriority()`** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.246.4.98 `solverState()`** [Answer](#) `smtrat::Module::solverState ( ) const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.246.4.99 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.246.4.100 `threadPriority()`** `thread_priority` `smtrat::Module::threadPriority () const [inline], [inherited]`

**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.246.4.101 `updateAllModels()`** `void smtrat::Module::updateAllModels () [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in `smtrat::SATModule< Settings >`.

**0.15.246.4.102 `updateLemmas()`** `void smtrat::Module::updateLemmas () [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.246.4.103 `updateModel()`** `void smtrat::PModule::updateModel () const [virtual], [inherited]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from `smtrat::Module`.

**0.15.246.4.104 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends () const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.246.5 Field Documentation

**0.15.246.5.1 `mAllBackends`** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.246.5.2 `mAllModels`** `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

**0.15.246.5.3 `mBackendsFoundAnswer`** `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.246.5.4 `mConstraintsToInform`** `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

**0.15.246.5.5 `mFinalCheck`** `bool smtrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.246.5.6 mFirstPosInLastBranches** `std::size_t smrat::Module::mFirstPosInLastBranches = 0`  
[static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.246.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.246.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.246.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.246.5.10 mFullCheck** `bool smrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.246.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smrat::Module::mInfeasibleSubsets`  
[protected], [inherited]  
Stores the infeasible subsets.

**0.15.246.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints`  
[protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.246.5.13 mLastBranches** `std::vector< Branching > smrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static],  
[inherited]  
Stores the last branches in a cycle buffer.

**0.15.246.5.14 mLemmas** `std::vector<Lemma> smrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.246.5.15 mModel** `Model smrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.246.5.16 mModelComputed** `bool smrat::Module::mModelComputed` [mutable], [protected],  
[inherited]  
True, if the model has already been computed.

**0.15.246.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.246.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]

Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.246.5.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]

Reusable splitting variables.

**0.15.246.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]

A reference to the manager.

**0.15.246.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]

Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.246.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.246.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.246.5.24 mUsedBackends** std::vector<Module\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.246.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.247 smtrat::ICPModule< Settings > Class Template Reference

#include <ICPModule.h>

### Data Structures

- struct **comp**

*TypeDefs:*

- struct **formulaPtrComp**
- struct **linearVariable**
- struct **weights**

## Public Types

- `typedef carl::FastPointerMap< Poly *, weights > WeightMap`
- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

## Public Member Functions

- `std::string moduleName () const`
- `ICPModule (const ModuleInput *, Conditionals &, Manager *const =NULL)`  
`Constructors:`
- `~ICPModule ()`  
`Destructor:`
- `bool informCore (const FormulaT &)`  
`Informs the module about the given constraint.`
- `bool addCore (ModuleInput::const_iterator)`  
`The module has to take the given sub-formula of the received formula into account.`
- `void removeCore (ModuleInput::const_iterator)`  
`Removes everything related to the given sub-formula of the received formula.`
- `Answer checkCore ()`  
`Checks the received formula for consistency.`
- `void updateModel () const`  
`Updates the model, if the solver has detected the consistency of the received formula, beforehand.`
- `EvalRationalIntervalMap getCurrentBoxAsIntervals () const`
- `FormulasT getCurrentBoxAsFormulas () const`
- `bool inform (const FormulaT &_constraint)`  
`Informs the module about the given constraint.`
- `void deinform (const FormulaT &_constraint)`  
`The inverse of informing about a constraint.`
- `virtual void init ()`  
`Informs all backends about the so far encountered constraints, which have not yet been communicated.`
- `bool add (ModuleInput::const_iterator _subformula)`  
`The module has to take the given sub-formula of the received formula into account.`
- `virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_← VARIABLE)`  
`Checks the received formula for consistency.`
- `virtual void remove (ModuleInput::const_iterator _subformula)`  
`Removes everything related to the given sub-formula of the received formula.`
- `virtual void updateAllModels ()`  
`Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.`
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`  
`Partition the variables from the current model into equivalence classes according to their assigned value.`
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`  
`Sets this modules unique ID to identify itself.`
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`  
`Sets the priority of this module to get a thread for running its check procedure.`
- `const ModuleInput * pReceivedFormula () const`

- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const `std::vector< Model > & allModels () const`
- const `std::vector< FormulaSetT > & infeasibleSubsets () const`
- const `std::vector< Module * > & usedBackends () const`
- const `carl::FastSet< FormulaT > & constraintsToInform () const`
- const `carl::FastSet< FormulaT > & informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const `std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `hasValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, FormulaT > `getReceivedFormulaSimplified ()`
- void `print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5

*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< `Branching` > `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))

*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0

*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< `FormulaT` > `mOldSplittingVariables`

*Reusable splitting variables.*

## Protected Member Functions

- `ModuleInput::iterator eraseSubformulaFromPassedFormula` (`ModuleInput::iterator` \_subformula, bool \_← ignoreOrigins=false)

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)

*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const

*Clears the assignment, if any was found.*
- void `clearModels` () const

*Clears all assignments, if any was found.*
- void `cleanModel` () const

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin` ()
- `ModuleInput::iterator passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const

*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_← origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)

*Adds the given formula to the passed formula with no origin.*

- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
- void [getInfeasibleSubsets](#) ()
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void [getBackendsModel](#) () const
- void [getBackendsAllModels](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
- virtual [Answer](#) [runBackends](#) (bool \_final, bool \_full, carl::Variable \_objective)
 

*Runs the backend solvers on the passed formula.*
- virtual [Answer](#) [runBackends](#) ()
 

*Runs the backend solvers on the passed formula.*
- void [clearPassedFormula](#) ()
 

*Merges the two vectors of sets into the first one.*
- size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
- bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const [Rational](#) &\_branchingValue) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- void [splitUnequalConstraint](#) (const [FormulaT](#) &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned [checkModel](#) () const
- void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()
 

*Gets all InformationRelevantFormulas.*
- bool [isLemmaLevel](#) ([LemmaLevel](#) level)
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector<FormulaSetT> mInfeasibleSubsets`

*Stores the infeasible subsets.*

- `Manager *const mpManager`

*A reference to the manager.*

- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*

- `std::vector<Model> mAllModels`

*Stores all satisfying assignments.*

- `bool mModelComputed`

*True, if the model has already been computed.*

- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*

- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*

- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*

- `std::vector<TheoryPropagation> mTheoryPropagations`

- `std::atomic<Answer> mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- `Conditionals mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- `std::vector<Module *> mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- `std::vector<Module *> mAllBackends`

*The backends of this module which have been used.*

- `std::vector<Lemma> mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*

- `ModuleInput::iterator mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- `carl::FastSet<FormulaT> mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet<FormulaT> mInformedConstraints`

*Stores the position of the first constraint of which no backend has been informed about.*

- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- `unsigned mSmallerMusesCheckCounter`

*Counter used for the generation of the smt2 files to check for smaller muses.*

- `std::vector<std::size_t> mVariableCounters`

*Maps variables to the number of their occurrences.*

### 0.15.247.1 Member Typedef Documentation

#### 0.15.247.1.1 SettingsType template<class Settings >

```
typedef Settings smtrat::ICPModule< Settings >::SettingsType
```

#### 0.15.247.1.2 WeightMap template<class Settings >

```
typedef carl::FastPointerMap<Poly*, weights> smtrat::ICPModule< Settings >::WeightMap
```

### 0.15.247.2 Member Enumeration Documentation

#### 0.15.247.2.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.247.3 Constructor & Destructor Documentation

#### 0.15.247.3.1 ICPModule() template<class Settings >

```
smtrat::ICPModule< Settings >::ICPModule (
 const ModuleInput * ,
 Conditionals & ,
 Manager * const = NULL)
```

Constructors:

#### 0.15.247.3.2 ~ICPModule() template<class Settings >

```
smtrat::ICPModule< Settings >::~ICPModule ()
```

Destructor:

### 0.15.247.4 Member Function Documentation

#### 0.15.247.4.1 add() bool smtrat::Module::add (

```
ModuleInput::const_iterator _subformula) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.247.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (``const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.247.4.3 addCore()** `template<class Settings >``bool smtrat::ICPModule< Settings >::addCore (``ModuleInput::const_iterator formula ) [virtual]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.247.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula (``const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.247.4.5 addLemma()** `void smtrat::Module::addLemma (``const FormulaT & _lemma,``const LemmaType & _lt = LemmaType::NORMAL,``const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],``[inherited]`

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.247.4.6 addOrigin()** `void smtrat::Module::addOrigin (``ModuleInput::iterator _formula,`

```
const FormulaT & _origin) [inline], [protected], [inherited]
Adds the given set of formulas in the received formula to the origins of the given passed formula.
```

#### Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.247.4.7 addReceivedSubformulaToPassedFormula()** std::pair<[ModuleInput::iterator](#),bool> smrat::Module::addReceivedSubformulaToPassedFormula (

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.247.4.8 addSubformulaToPassedFormula() [1/3]** std::pair<[ModuleInput::iterator](#),bool> smrat::Module::addSubformulaToPassedFormula (

```
const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.247.4.9 addSubformulaToPassedFormula() [2/3]** std::pair<[ModuleInput::iterator](#),bool> smrat::Module::addSubformulaToPassedFormula (

```
const FormulaT & _formula,
```

```
const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.247.4.10 addSubformulaToPassedFormula() [3/3]** `std::pair<ModuleInput::iterator, bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.247.4.11 allModels()** `const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]`**Returns**

All satisfying assignments, if existent.

**0.15.247.4.12 anAnswerFound()** `bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.247.4.13 answerFound()** `const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]`**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.247.4.14 backendsModel()** `const Model & smrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.247.4.15 branchAt() [1/4]** bool smtrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.247.4.16 branchAt() [2/4]** bool smtrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.247.4.17 branchAt() [3/4]** bool smtrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.247.4.18 branchAt() [4/4]** bool smtrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

---

**0.15.247.4.19 check()** *Answer* smtrat::Module::check (

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.247.4.20 checkCore()** *template<class Settings >*  
*Answer* smtrat::ICPModule< Settings >::checkCore ( ) [virtual]

Checks the received formula for consistency.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.247.4.21 checkInfSubsetForMinimality()** *void* smtrat::Module::checkInfSubsetForMinimality (

```
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.247.4.22 checkModel()** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.247.4.23 cleanModel()** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.247.4.24 clearLemmas()** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.**0.15.247.4.25 clearModel()** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.247.4.26 clearModels()** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.247.4.27 clearPassedFormula()** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`**0.15.247.4.28 collectOrigins() [1/2]** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`**0.15.247.4.29 collectOrigins() [2/2]** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.247.4.30 collectTheoryPropagations()** `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

**0.15.247.4.31 constraintsToInform()** const carl::FastSet<[FormulaT](#)>& smrat::Module::constraintsToInform( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.247.4.32 currentlySatisfied()** virtual unsigned smrat::Module::currentlySatisfied( const [FormulaT](#) & ) const [inline], [virtual], [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.247.4.33 currentlySatisfiedByBackend()** unsigned smrat::Module::currentlySatisfiedByBackend( const [FormulaT](#) & \_formula ) const [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.247.4.34 deinform()** void smrat::Module::deinform( const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.247.4.35 deinformCore()** virtual void smrat::Module::deinformCore( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.247.4.36 determine\_smallest\_origin()** size\_t smrat::Module::determine\_smallest\_origin( const std::vector< [FormulaT](#) > & origins ) const [protected], [inherited]

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>origins</i> | A vector of sets of origins. |
|----------------|------------------------------|

## Returns

The index of the smallest (regarding the size of the sets) element of origins

```
0.15.247.4.37 eraseSubformulaFromPassedFormula() template<class Settings >
ModuleInput::iterator smtrat::ICPModule< Settings >::eraseSubformulaFromPassedFormula (
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
Afterwards the sub-formula is removed from the passed formula.

## Parameters

|                       |                                                                                                             |
|-----------------------|-------------------------------------------------------------------------------------------------------------|
| <i>_subformula</i>    | The sub-formula to remove from the passed formula.                                                          |
| <i>_ignoreOrigins</i> | SAT, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

## Returns

Reimplemented from [smtrat::Module](#).

```
0.15.247.4.38 excludeNotReceivedVariablesFromModel() void smtrat::Module::excludeNotReceived←
VariablesFromModel () const [inherited]
Excludes all variables from the current model, which do not occur in the received formula.
```

```
0.15.247.4.39 findBestOrigin() std::vector< FormulaT >::const_iterator smtrat::Module::find←
BestOrigin (
 const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

## Parameters

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

## Returns

```
0.15.247.4.40 firstSubformulaToPass() ModuleInput::const_iterator smtrat::Module::firstSubformula←
ToPass () const [inline], [inherited]
```

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.247.4.41 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.247.4.42 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.247.4.43 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.247.4.44 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.247.4.45 getBackendsModel()** `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.247.4.46 getCurrentBoxAsFormulas()** `template<class Settings > FormulasT smtrat::ICPModule< Settings >::getCurrentBoxAsFormulas ( ) const`

**0.15.247.4.47 getCurrentBoxAsIntervals()** `template<class Settings > EvalRationalIntervalMap smtrat::ICPModule< Settings >::getCurrentBoxAsIntervals ( ) const`

**0.15.247.4.48 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.247.4.49 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.247.4.50 getInformationRelevantFormulas()** const std::set< [FormulaT](#) > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.247.4.51 getModelEqualities()** std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.247.4.52 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const [FormulaT](#) & *\_formula*, [FormulaSetT](#) & *\_origins* ) const [inline], [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_formula</i> |  |
| <i>_origins</i> |  |

**0.15.247.4.53 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const [FormulaT](#) & *\_formula*, [FormulasT](#) & *\_origins* ) const [inline], [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_formula</i> |  |
| <i>_origins</i> |  |

**0.15.247.4.54 getOrigins() [3/3]** const [FormulaT](#)& smtrat::Module::getOrigins ( [ModuleInput::const\\_iterator](#) *\_formula* ) const [inline], [protected], [inherited]  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.247.4.55 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified( ) [virtual], [inherited]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.247.4.56 `hasLemmas()`** `bool smtrat::Module::hasLemmas( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

**0.15.247.4.57 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.247.4.58 `id()`** `std::size_t smtrat::Module::id( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.247.4.59 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.247.4.60 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.247.4.61 informBackends()** void smtrat::Module::informBackends (

```
const FormulaT & _constraint) [inline], [protected], [inherited]
```

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.247.4.62 informCore()** template<class Settings >

```
bool smtrat::ICPModule< Settings >::informCore (
 const FormulaT & _constraint) [virtual]
```

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.247.4.63 informedConstraints()** const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.247.4.64 init()** void smtrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smtrat::PBPPModule](#)< Settings >, [smtrat::PBGaussModule](#)< Settings >, [smtrat::NRAILModule](#)< Settings >, [smtrat::NewCoveringModule](#)< Settings >, [smtrat::NewCADModule](#)< Settings >, [smtrat::LRAModule](#)< Settings >, [smtrat::LRAModule](#)< LRASettingsICP >, [smtrat::LRAModule](#)< LRASettings1 >, [smtrat::IntBlastModule](#)< Settings >, [smtrat::FPPModule](#)< Settings >, [smtrat::EQModule](#)< Settings >, [smtrat::CurryModule](#)< Settings >, and [smtrat::BVMModule](#)< Settings >.

**0.15.247.4.65 is\_minimizing()** bool smtrat::Module::is\_minimizing ( ) const [inline], [inherited]**0.15.247.4.66 isLemmaLevel()** bool smtrat::Module::isLemmaLevel (

```
LemmaLevel level) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.247.4.67 `isPreprocessor()`** `bool smtrat::Module::isPreprocessor () const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.247.4.68 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas () const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.247.4.69 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.247.4.70 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.247.4.71 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                |                                |
|----------------|--------------------------------|
| <i>_modelA</i> | The first model to check for.  |
| <i>_modelB</i> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.247.4.72 `moduleName()`** `template<class Settings >`  
`std::string smtrat::ICPModule< Settings >::moduleName ( ) const [inline], [virtual]`

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.247.4.73 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.247.4.74 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`  
`const FormulaT & _origin ) const [protected], [inherited]`

**0.15.247.4.75 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin (`  
`) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.247.4.76 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
`[inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.247.4.77 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.247.4.78 `pReceivedFormula()`** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.247.4.79 `print()`** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.247.4.80 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.247.4.81 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.247.4.82 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.247.4.83 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.247.4.84 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.247.4.85 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.247.4.86 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline], [inherited]
```

Notifies that the received formulas has been checked.

```
0.15.247.4.87 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs<= InfeasibleSubset (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.247.4.88 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.247.4.89 remove() void smtrat::Module::remove (
```

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub formula of the received formula to remove. |
|--------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

```
0.15.247.4.90 removeCore() template<class Settings >
void smtrat::ICPModule< Settings >::removeCore (
```

```
 ModuleInput::const_iterator formula) [virtual]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>formula</i> | The sub formula of the received formula to remove. |
|----------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.247.4.91 removeOrigin()** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.247.4.92 removeOrigins()** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.247.4.93 rPassedFormula()** const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.247.4.94 rReceivedFormula()** const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.247.4.95 runBackends() [1/2]** virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]

Reimplemented in [smtrat::PModule](#).

**0.15.247.4.96 runBackends() [2/2]** Answer smtrat::Module::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.247.4.97 setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]  
 Sets this modules unique ID to identify itself.

#### Parameters

|                                                                                            |                                    |
|--------------------------------------------------------------------------------------------|------------------------------------|
| <br>$_id$ | The id to set this module's id to. |
|--------------------------------------------------------------------------------------------|------------------------------------|

**0.15.247.4.98 setThreadPriority()** void smtrat::Module::setThreadPriority ( thread\_priority \_threadPriority ) [inline], [inherited]  
 Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| $_threadPriority$ | The priority to set this module's thread priority to. |
|-------------------|-------------------------------------------------------|

**0.15.247.4.99 solverState()** Answer smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.247.4.100 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]  
 Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                      |                                                  |
|----------------------|--------------------------------------------------|
| $_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|----------------------|--------------------------------------------------|

**0.15.247.4.101 threadPriority()** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.247.4.102 updateAllModels()** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]  
 Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
 Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.247.4.103 updateLemmas()** void smrat::Module::updateLemmas ( ) [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.247.4.104 updateModel()** template<class Settings >  
void smrat::ICPModule< Settings >::updateModel ( ) const [virtual]  
Updates the model, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented from [smrat::Module](#).

**0.15.247.4.105 usedBackends()** const std::vector<[Module](#)\*>& smrat::Module::usedBackends ( )  
const [inline], [inherited]

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.247.5 Field Documentation

**0.15.247.5.1 mAllBackends** std::vector<[Module](#)\*> smrat::Module::mAllBackends [protected],  
[inherited]  
The backends of this module which have been used.

**0.15.247.5.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected],  
[inherited]  
Stores all satisfying assignments.

**0.15.247.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer  
[protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.247.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform  
[protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.247.5.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.247.5.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.247.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.247.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.247.5.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.247.5.10 mFullCheck** `bool smtrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.247.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`  
Stores the infeasible subsets.

**0.15.247.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.247.5.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`  
Stores the last branches in a cycle buffer.

**0.15.247.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.247.5.15 mModel** `Model smtrat::Module::mModel [mutable], [protected], [inherited]`  
Stores the assignment of the current satisfiable result, if existent.

**0.15.247.5.16 mModelComputed** `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`  
True, if the model has already been computed.

**0.15.247.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.247.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.247.5.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables [static], [inherited]`  
Reusable splitting variables.

**0.15.247.5.20 mpManager** `Manager* const smtrat::Module::mpManager [protected], [inherited]`  
A reference to the manager.

**0.15.247.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.247.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.247.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

**0.15.247.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.247.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`  
Maps variables to the number of their occurrences.

## 0.15.248 smtrat::icp::IcpVariable Class Reference

```
#include <IcpVariable.h>
```

### Public Member Functions

- `IcpVariable (carl::Variable::Arg _var, bool _original, ModuleInput::iterator _defaultPosition, EvalDoubleIntervalMap::iterator _intervalPos, const LRAVariable *_lraVar=NULL)`
- `~IcpVariable ()`
- `carl::Variable::Arg var () const`
- `ContractionCandidates & candidates ()`
- `const LRAVariable * lraVar () const`
- `void addCandidate (ContractionCandidate *_candidate)`
- `void addCandidates (const ContractionCandidates &_candidates)`
- `void addOriginalConstraint (const FormulaT &_constraint)`
- `void removeOriginalConstraint (const FormulaT &_constraint)`
- `void setLraVar (const LRAVariable *_lraVar)`
- `void print (std::ostream &_out=std::cout, bool _withContractionCandidates=false) const`
- `bool isActive () const`
- `void incrementActivity ()`
- `void decrementActivity ()`
- `bool is_linear ()`
- `EvalDoubleIntervalMap::const_iterator intervalPosition () const`
- `const DoubleInterval & interval () const`
- `void setInterval (const DoubleInterval &_interval)`
- `void setUnmodified ()`
- `void setExternalUnmodified ()`
- `void setExternalModified ()`

- void `setInternalUnmodified ()`
- `Updated isInternalUpdated () const`
- `Updated isExternalUpdated () const`
- const `smtrat::FormulaT & internalLeftBound () const`
- const `smtrat::FormulaT & internalRightBound () const`
- `ModuleInput::iterator externalLeftBound () const`
- `ModuleInput::iterator externalRightBound () const`
- void `setInternalLeftBound (const smtrat::FormulaT &_left)`
- void `setInternalRightBound (const smtrat::FormulaT &_right)`
- void `setExternalLeftBound (ModuleInput::iterator _left)`
- void `setExternalRightBound (ModuleInput::iterator _right)`
- `Updated isInternalBoundsSet () const`
- bool `isOriginal () const`
- bool `operator< (IcpVariable const &rhs) const`

**Friends**

- std::ostream & `operator<< (std::ostream &os, const IcpVariable &_var)`

**0.15.248.1 Constructor & Destructor Documentation**

**0.15.248.1.1 `IcpVariable()`** smtrat::icp::IcpVariable::IcpVariable ( carl::Variable::Arg \_var,  
bool \_original,  
ModuleInput::iterator \_defaultPosition,  
EvalDoubleIntervalMap::iterator \_intervalPos,  
const LRAVariable \* \_lraVar = NULL ) [inline]

**0.15.248.1.2 `~IcpVariable()`** smtrat::icp::IcpVariable::~IcpVariable ( ) [inline]

**0.15.248.2 Member Function Documentation**

**0.15.248.2.1 `addCandidate()`** void smtrat::icp::IcpVariable::addCandidate ( ContractionCandidate \* \_candidate ) [inline]

**0.15.248.2.2 `addCandidates()`** void smtrat::icp::IcpVariable::addCandidates ( const ContractionCandidates & \_candidates ) [inline]

**0.15.248.2.3 `addOriginalConstraint()`** void smtrat::icp::IcpVariable::addOriginalConstraint ( const FormulaT & \_constraint ) [inline]

**0.15.248.2.4 `candidates()`** ContractionCandidates& smtrat::icp::IcpVariable::candidates ( ) [inline]

**0.15.248.2.5 `decrementActivity()`** void smtrat::icp::IcpVariable::decrementActivity ( ) [inline]

**0.15.248.2.6 `externalLeftBound()`** `ModuleInput::iterator` `smtrat::icp::IcpVariable::externalLeftBound( ) const` [inline]

**0.15.248.2.7 `externalRightBound()`** `ModuleInput::iterator` `smtrat::icp::IcpVariable::externalRightBound( ) const` [inline]

**0.15.248.2.8 `incrementActivity()`** `void smtrat::icp::IcpVariable::incrementActivity( )` [inline]

**0.15.248.2.9 `internalLeftBound()`** `const smtrat::FormulaT&` `smtrat::icp::IcpVariable::internalLeftBound( ) const` [inline]

**0.15.248.2.10 `internalRightBound()`** `const smtrat::FormulaT&` `smtrat::icp::IcpVariable::internalRightBound( ) const` [inline]

**0.15.248.2.11 `interval()`** `const DoubleInterval&` `smtrat::icp::IcpVariable::interval( ) const` [inline]

**0.15.248.2.12 `intervalPosition()`** `EvalDoubleIntervalMap::const_iterator` `smtrat::icp::IcpVariable::intervalPosition( ) const` [inline]

**0.15.248.2.13 `is_linear()`** `bool smtrat::icp::IcpVariable::is_linear( )` [inline]

**0.15.248.2.14 `isActive()`** `bool smtrat::icp::IcpVariable::isActive( ) const` [inline]

**0.15.248.2.15 `isExternalUpdated()`** `Updated smtrat::icp::IcpVariable::isExternalUpdated( ) const` [inline]

**0.15.248.2.16 `isInternalBoundsSet()`** `Updated smtrat::icp::IcpVariable::isInternalBoundsSet( ) const` [inline]

**0.15.248.2.17 `isInternalUpdated()`** `Updated smtrat::icp::IcpVariable::isInternalUpdated( ) const` [inline]

**0.15.248.2.18 `isOriginal()`** `bool smtrat::icp::IcpVariable::isOriginal( ) const` [inline]

**0.15.248.2.19 `lraVar()`** `const LRAVariable* smtrat::icp::IcpVariable::lraVar( ) const` [inline]

**0.15.248.2.20 `operator<()`** `bool smtrat::icp::IcpVariable::operator<( IcpVariable const & rhs ) const` [inline]

- 0.15.248.2.21 `print()`** void smtrat::icp::IcpVariable::print ( std::ostream & \_out = std::cout, bool \_withContractionCandidates = false ) const [inline]
- 0.15.248.2.22 `removeOriginalConstraint()`** void smtrat::icp::IcpVariable::removeOriginalConstraint ( const FormulaT & \_constraint ) [inline]
- 0.15.248.2.23 `setExternalLeftBound()`** void smtrat::icp::IcpVariable::setExternalLeftBound ( ModuleInput::iterator \_left ) [inline]
- 0.15.248.2.24 `setExternalModified()`** void smtrat::icp::IcpVariable::setExternalModified ( ) [inline]
- 0.15.248.2.25 `setExternalRightBound()`** void smtrat::icp::IcpVariable::setExternalRightBound ( ModuleInput::iterator \_right ) [inline]
- 0.15.248.2.26 `setExternalUnmodified()`** void smtrat::icp::IcpVariable::setExternalUnmodified ( ) [inline]
- 0.15.248.2.27 `setInternalLeftBound()`** void smtrat::icp::IcpVariable::setInternalLeftBound ( const smtrat::FormulaT & \_left ) [inline]
- 0.15.248.2.28 `setInternalRightBound()`** void smtrat::icp::IcpVariable::setInternalRightBound ( const smtrat::FormulaT & \_right ) [inline]
- 0.15.248.2.29 `setInternalUnmodified()`** void smtrat::icp::IcpVariable::setInternalUnmodified ( ) [inline]
- 0.15.248.2.30 `setInterval()`** void smtrat::icp::IcpVariable::setInterval ( const DoubleInterval & \_interval ) [inline]
- 0.15.248.2.31 `setLraVar()`** void smtrat::icp::IcpVariable::setLraVar ( const LRAVariable \* \_lraVar ) [inline]
- 0.15.248.2.32 `setUnmodified()`** void smtrat::icp::IcpVariable::setUnmodified ( ) [inline]
- 0.15.248.2.33 `var()`** carl::Variable::Arg smtrat::icp::IcpVariable::var ( ) const [inline]

### 0.15.248.3 Friends And Related Function Documentation

```
0.15.248.3.1 operator<< std::ostream& operator<< (
 std::ostream & os,
 const IcpVariable & _var) [friend]
```

## 0.15.249 smtrat::icp::icpVariableComp Struct Reference

```
#include <IcpVariable.h>
```

### Public Member Functions

- bool `operator()` (const `IcpVariable` \*const `_lhs`, const `IcpVariable` \*const `_rhs`) const

### 0.15.249.1 Member Function Documentation

```
0.15.249.1.1 operator()() bool smtrat::icp::icpVariableComp::operator() (
 const IcpVariable *const _lhs,
 const IcpVariable *const _rhs) const [inline]
```

## 0.15.250 smtrat::parser::Identifier Struct Reference

```
#include <Common.h>
```

### Public Member Functions

- `Identifier ()`
- `Identifier (const std::string &symbol)`
- `Identifier (const std::string &symbol, const std::vector< std::size_t > &indices)`
- `Identifier (const std::string &symbol, const std::vector< Integer > &indices)`
- `Identifier & operator= (const Identifier &i)`
- `Identifier (const Identifier &i)`
- `~Identifier ()`
- `operator std::string () const`

### Data Fields

- `std::string symbol`
- `std::vector< std::size_t > * indices`

### 0.15.250.1 Constructor & Destructor Documentation

```
0.15.250.1.1 Identifier() [1/5] smtrat::parser::Identifier::Identifier () [inline]
```

```
0.15.250.1.2 Identifier() [2/5] smtrat::parser::Identifier::Identifier (
 const std::string & symbol) [inline]
```

```
0.15.250.1.3 Identifier() [3/5] smtrat::parser::Identifier::Identifier (
 const std::string & symbol,
 const std::vector< std::size_t > & indices) [inline]
```

**0.15.250.1.4 Identifier()** [4/5] smtrat::parser::Identifier::Identifier ( const std::string & symbol, const std::vector< Integer > & indices ) [inline]

**0.15.250.1.5 Identifier()** [5/5] smtrat::parser::Identifier::Identifier ( const Identifier & i ) [inline]

**0.15.250.1.6 ~Identifier()** smtrat::parser::Identifier::~Identifier ( ) [inline]

## 0.15.250.2 Member Function Documentation

**0.15.250.2.1 operator std::string()** smtrat::parser::Identifier::operator std::string ( ) const [inline]

**0.15.250.2.2 operator=()** Identifier& smtrat::parser::Identifier::operator= ( const Identifier & i ) [inline]

## 0.15.250.3 Field Documentation

**0.15.250.3.1 indices** std::vector<std::size\_t>\* smtrat::parser::Identifier::indices

**0.15.250.3.2 symbol** std::string smtrat::parser::Identifier::symbol

## 0.15.251 smtrat::parser::IdentifierParser Struct Reference

#include <Identifier.h>

### Public Member Functions

- IdentifierParser ()

### Data Fields

- SymbolParser symbol
- NumeralParser numeral
- qi::rule< Iterator, Identifier(), Skipper > main
- qi::rule< Iterator, Identifier(), Skipper > indexed

## 0.15.251.1 Constructor & Destructor Documentation

**0.15.251.1.1 IdentifierParser()** smtrat::parser::IdentifierParser::IdentifierParser ( ) [inline]

## 0.15.251.2 Field Documentation

**0.15.251.2.1 indexed** `qi::rule<Iterator, Identifier(), Skipper> smtrat::parser::Identifier<Parser::indexed`

**0.15.251.2.2 main** `qi::rule<Iterator, Identifier(), Skipper> smtrat::parser::Identifier<Parser::main`

**0.15.251.2.3 numeral** `NumeralParser smtrat::parser::IdentifierParser::numeral`

**0.15.251.2.4 symbol** `SymbolParser smtrat::parser::IdentifierParser::symbol`

## 0.15.252 smtrat::cad::debug::IDSanitizer Struct Reference

```
#include <TikzHistoryPrinter.h>
```

### Public Member Functions

- `IDSanitizer` (const std::string &prefix)
- `std::string operator()` (const std::string &raw)

### 0.15.252.1 Constructor & Destructor Documentation

**0.15.252.1.1 IDSanitizer()** `smtrat::cad::debug::IDSanitizer::IDSanitizer ( const std::string & prefix ) [inline]`

### 0.15.252.2 Member Function Documentation

**0.15.252.2.1 operator()** `std::string smtrat::cad::debug::IDSanitizer::operator() ( const std::string & raw ) [inline]`

## 0.15.253 smtrat::mcsat::fm::IgnoreCoreSettings Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr bool `use_all_constraints` = true

### 0.15.253.1 Field Documentation

**0.15.253.1.1 use\_all\_constraints** `constexpr bool smtrat::mcsat::fm::IgnoreCoreSettings::use_all<_constraints = true [static], [constexpr]`

## 0.15.254 smtrat::IncWidthModule< Settings > Class Template Reference

```
#include <IncWidthModule.h>
```

## Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

## Public Member Functions

- `std::string moduleName () const`
- `IncWidthModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~IncWidthModule ()`
- `bool addCore (ModuleInput::const_iterator _subformula)`  
`The module has to take the given sub-formula of the received formula into account.`
- `void removeCore (ModuleInput::const_iterator _subformula)`  
`Removes the subformula of the received formula at the given position to the considered ones of this module.`
- `void updateModel () const`  
`Updates the current assignment into the model.`
- `Answer checkCore ()`  
`Checks the received formula for consistency.`
- `bool inform (const FormulaT &_constraint)`  
`Informs the module about the given constraint.`
- `void deform (const FormulaT &_constraint)`  
`The inverse of informing about a constraint.`
- `virtual void init ()`  
`Informs all backends about the so far encountered constraints, which have not yet been communicated.`
- `bool add (ModuleInput::const_iterator _subformula)`  
`The module has to take the given sub-formula of the received formula into account.`
- `virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
`Checks the received formula for consistency.`
- `virtual void remove (ModuleInput::const_iterator _subformula)`  
`Removes everything related to the given sub-formula of the received formula.`
- `virtual void updateAllModels ()`  
`Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.`
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`  
`Partition the variables from the current model into equivalence classes according to their assigned value.`
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`  
`Sets this modules unique ID to identify itself.`
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`  
`Sets the priority of this module to get a thread for running its check procedure.`
- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`
- `const std::vector< Model > & allModels () const`
- `const std::vector< FormulaSetT > & infeasibleSubsets () const`
- `const std::vector< Module * > & usedBackends () const`

- const carl::FastSet< `FormulaT` > & `constraintsToInform` () const
- const carl::FastSet< `FormulaT` > & `informedConstraints` () const
- void `addLemma` (const `FormulaT` &\_lemma, const `LemmaType` &\_lt=`LemmaType::NORMAL`, const `FormulaT` &\_preferredFormula=`FormulaT(carl::FormulaType::TRUE)`)
 

*Stores a lemma being a valid formula.*
- bool `hasLemmas` ()
 

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas` ()
 

*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas` () const
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula` () const
- `ModuleInput::const_iterator firstSubformulaToPass` () const
- void `receivedFormulaChecked` ()
 

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals` & `answerFound` () const
- bool `isPreprocessor` () const
- carl::Variable `objective` () const
- bool `is_minimizing` () const
- void `excludeNotReceivedVariablesFromModel` () const
 

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas` ()
 

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations` ()
- void `collectOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
 

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- bool `hasValidInfeasibleSubset` () const
- void `checkInfSubsetForMinimality` (std::vector< `FormulaSetT` >::const\_iterator \_infsSubset, const std::string &\_filename="smaller\_muses", unsigned \_maxSizeDifference=1) const
 

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, `FormulaT` > `getReceivedFormulaSimplified` ()
- void `print` (const std::string &\_initiation="\*\*\*") const
 

*Prints everything relevant of the solver.*
- void `printReceivedFormula` (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of the received formula.*
- void `printPassedFormula` (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const
 

*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const
 

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)
 

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const FormulaT &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin` ()
- `ModuleInput::iterator passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const FormulaT &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const FormulaT & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const FormulaT &\_formula, FormulasT &\_origins) const
- void `getOrigins` (const FormulaT &\_formula, FormulaSetT &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const FormulaT &\_origin)  
*Copies the given sub-formula of the received formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)
- void `informBackends` (const FormulaT &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const FormulaT &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)  
*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula)  
*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)  
*Adds the given formula to the passed formula.*

- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &`_formula`, const `FormulaT` &`_origin`)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` `_subformula`)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &`_origins`) const
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &`_backend`) const
 

*Get the infeasible subsets the given backend provides.*
- const `Model` & `backendsModel` () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel` () const
 

*Stores all models of a backend in the list of all models of this module.*
- void `getBackendsAllModels` () const
 

*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends` (bool `_final`, bool `_full`, carl::Variable `_objective`)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- virtual `Answer runBackends` ()
 

*Merges the two vectors of sets into the first one.*
- std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const `Poly` &`_polynomial`, bool `_integral`, const `Rational` &`_value`, std::vector< `FormulaT` > &&`_premise`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (const `Poly` &`_polynomial`, bool `_integral`, const `Rational` &`_value`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`, const std::vector< `FormulaT` > &`_premise=std::vector< FormulaT >()`)
 

*Adds a formula to the InformationRelevantFormula.*
- bool `branchAt` (carl::Variable::Arg `_var`, const `Rational` &`_value`, std::vector< `FormulaT` > &&`_premise`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`)
 

*Adds a formula to the InformationRelevantFormula.*
- bool `branchAt` (carl::Variable::Arg `_var`, const `Rational` &`_value`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`, const std::vector< `FormulaT` > &`_premise=std::vector< FormulaT >()`)
 

*Adds the following lemmas for the given constraint  $p!=0$ .*
- unsigned `checkModel` () const
 

*Gets all InformationRelevantFormulas.*
- void `addInformationRelevantFormula` (const `FormulaT` &`formula`)
 

*Checks if current lemma level is greater or equal to given level.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector<FormulaSetT> mInfeasibleSubsets`

*Stores the infeasible subsets.*

- `Manager *const mpManager`

*A reference to the manager.*

- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*

- `std::vector<Model> mAllModels`

*Stores all satisfying assignments.*

- `bool mModelComputed`

*True, if the model has already been computed.*

- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*

- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*

- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*

- `std::vector<TheoryPropagation> mTheoryPropagations`

- `std::atomic<Answer> mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- `Conditionals mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- `std::vector<Module *> mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- `std::vector<Module *> mAllBackends`

*The backends of this module which have been used.*

- `std::vector<Lemma> mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*

- `ModuleInput::iterator mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- `carl::FastSet<FormulaT> mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet<FormulaT> mInformedConstraints`

*Stores the position of the first constraint of which no backend has been informed about.*

- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- `unsigned mSmallerMusesCheckCounter`

*Counter used for the generation of the smt2 files to check for smaller muses.*

- `std::vector<std::size_t> mVariableCounters`

*Maps variables to the number of their occurrences.*

### 0.15.254.1 Member Typedef Documentation

**0.15.254.1.1 SettingsType** template<typename Settings >  
typedef `Settings smtrat::IncWidthModule< Settings >::SettingsType`

### 0.15.254.2 Member Enumeration Documentation

**0.15.254.2.1 LemmaType** enum `smtrat::Module::LemmaType` : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.254.3 Constructor & Destructor Documentation

**0.15.254.3.1 IncWidthModule()** template<typename Settings >  
`smtrat::IncWidthModule< Settings >::IncWidthModule` (  
    const `ModuleInput` \* `_formula`,  
    `Conditionals` & `_conditionals`,  
    `Manager` \* `_manager` = `NULL` )

**0.15.254.3.2 ~IncWidthModule()** template<typename Settings >  
`smtrat::IncWidthModule< Settings >::~IncWidthModule` ( )

### 0.15.254.4 Member Function Documentation

**0.15.254.4.1 add()** bool `smtrat::Module::add` (  
    `ModuleInput::const_iterator` `_subformula` ) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.254.4.2 addConstraintToInform()** void `smtrat::Module::addConstraintToInform` (  
    const `FormulaT` & `_constraint` ) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                    |                        |
|--------------------|------------------------|
| <i>_constraint</i> | The constraint to add. |
|--------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.254.4.3 addCore()** `template<typename Settings >`

```
bool smtrat::IncWidthModule< Settings >::addCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub-formula to take additionally into account. |
|--------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.254.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula (`

```
 const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.254.4.5 addLemma()** `void smtrat::Module::addLemma (`

```
 const FormulaT & _lemma,
```

```
 const LemmaType & _lt = LemmaType::NORMAL,
```

```
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
```

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.254.4.6 addOrigin()** `void smtrat::Module::addOrigin (`

```
 ModuleInput::iterator _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.254.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addReceivedSubformulaToPassedFormula (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.254.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.254.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.254.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.254.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline],
[inherited]
```

#### Returns

All satisfying assignments, if existent.

```
0.15.254.4.12 anAnswerFound() bool smrat::Module::anAnswerFound () const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

```
0.15.254.4.13 answerFound() const smrat::Conditionals& smrat::Module::answerFound () const
[inline], [inherited]
```

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

```
0.15.254.4.14 backendsModel() const Model & smrat::Module::backendsModel () const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.254.4.15** **branchAt()** [1/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.254.4.16** **branchAt()** [2/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.254.4.17** **branchAt()** [3/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.254.4.18** **branchAt()** [4/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.254.4.19 check() Answer smrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::PModule](#).

```
0.15.254.4.20 checkCore() template<typename Settings >
Answer smrat::IncWidthModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

```
0.15.254.4.21 checkInfeasibleSubsetForMinimality() void smrat::Module::checkInfeasibleSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.254.4.22 checkModel() unsigned smrat::Module::checkModel () const [protected], [inherited]
```

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.254.4.23 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.254.4.24 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.254.4.25 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.254.4.26 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.254.4.27 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.254.4.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.254.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.254.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.254.4.31 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.254.4.32 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.254.4.33 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.254.4.34 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.254.4.35 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.254.4.36 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.254.4.37 eraseSubformulaFromPassedFormula()** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

```
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.254.4.38 excludeNotReceivedVariablesFromModel()** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.254.4.39 findBestOrigin()** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`

```
 const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns****0.15.254.4.40 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.254.4.41 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.254.4.42 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.254.4.43 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.254.4.44 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.254.4.45 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.254.4.46 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.254.4.47 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.254.4.48 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.254.4.49 getModelEqualities()** std::list< std::vector< `carl::Variable` > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.254.4.50 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.254.4.51 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.254.4.52 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

**Parameters**

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.254.4.53 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.254.4.54 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.254.4.55 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.254.4.56 `id()`** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.254.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.254.4.58 `inform()`** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.254.4.59 `informBackends()`** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.254.4.60 `informCore()`** `virtual bool smtrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.254.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.254.4.62 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.254.4.63 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.254.4.64 isLemmaLevel()** `bool smrat::Module::isLemmaLevel( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.254.4.65 isPreprocessor()** `bool smrat::Module::isPreprocessor( ) const [inline], [inherited]`

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.254.4.66 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.254.4.67 merge()** `std::vector< FormulaT > smrat::Module::merge( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.254.4.68 `model()`** `const Model& smtrat::Module::model( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.254.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint( const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.254.4.70 `moduleName()`** `template<typename Settings> std::string smtrat::IncWidthModule< Settings >::moduleName( ) const [inline], [virtual]`**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.254.4.71 `objective()`** `carl::Variable smtrat::Module::objective( ) const [inline], [inherited]`**0.15.254.4.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula( const FormulaT & _origin ) const [protected], [inherited]`**0.15.254.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin( ) [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.254.4.74 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ()`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.254.4.75 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula () const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.254.4.76 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula () const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.254.4.77 print()** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const` [inherited]  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.254.4.78 printAllModels()** `void smtrat::Module::printAllModels (`  
`std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.254.4.79 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets (`  
`const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.254.4.80 printModel()** `void smtrat::Module::printModel (`  
`std::ostream & _out = std::cout ) const` [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.254.4.81 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.254.4.82 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.254.4.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.254.4.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.254.4.85 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAs<-InfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.254.4.86 receivedVariable()** `bool smrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.254.4.87 remove()** `void smrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

**0.15.254.4.88 removeCore()** `template<typename Settings > void smrat::IncWidthModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula ) [virtual]`

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.254.4.89 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.254.4.90 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

**0.15.254.4.91 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const [inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.254.4.92 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A reference to the formula passed to this module.

**0.15.254.4.93 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.254.4.94 runBackends() [2/2]** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.254.4.95 setId()** `void smtrat::Module::setId (`

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

**Parameters**

|                   |                                    |
|-------------------|------------------------------------|
| <code>↔ id</code> | The id to set this module's id to. |
|-------------------|------------------------------------|

**0.15.254.4.96 setThreadPriority()** `void smtrat::Module::setThreadPriority (`

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.254.4.97 solverState()** `Answer smtrat::Module::solverState () const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.254.4.98 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.254.4.99 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.254.4.100 updateAllModels()** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.254.4.101 updateLemmas()** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.254.4.102 updateModel()** `template<typename Settings >`

`void smtrat::IncWidthModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.254.4.103 usedBackends()** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.254.5 Field Documentation****0.15.254.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.254.5.2 mAllModels** `std::vector<Model>` `smtrat::Module::mAllModels` [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.254.5.3 mBackendsFoundAnswer** `std::atomic_bool*` `smtrat::Module::mBackendsFoundAnswer` [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.254.5.4 mConstraintsToInform** `carl::FastSet<FormulaT>` `smtrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.254.5.5 mFinalCheck** `bool` `smtrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.254.5.6 mFirstPosInLastBranches** `std::size_t` `smtrat::Module::mFirstPosInLastBranches` = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.254.5.7 mFirstSubformulaToPass** `ModuleInput::iterator` `smtrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.254.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator` `smtrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.254.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.254.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.254.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.254.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.254.5.13 mLastBranches** std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.254.5.14 mLemmas** std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]

Stores the lemmas being valid formulas this module or its backends made.

**0.15.254.5.15 mModel** Model smtrat::Module::mModel [mutable], [protected], [inherited]

Stores the assignment of the current satisfiable result, if existent.

**0.15.254.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]

True, if the model has already been computed.

**0.15.254.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore =

5 [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.254.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]

Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.254.5.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]

Reusable splitting variables.

**0.15.254.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]

A reference to the manager.

**0.15.254.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]

Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.254.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.254.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.254.5.24 mUsedBackends** std::vector<Module\*> smrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.254.5.25 mVariableCounters** std::vector<std::size\_t> smrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

**0.15.255 smrat::parser::IndexedFunctionInstantiator Struct Reference**

```
#include <FunctionInstantiator.h>
```

**Public Member Functions**

- virtual ~IndexedFunctionInstantiator ()
- template<typename T >  
  bool convert (const std::vector< types::TermType > &from, std::vector< T > &to) const
- virtual bool operator() (const std::vector< std::size\_t > &, const std::vector< types::TermType > &, types::TermType &, TheoryError &errors) const

**0.15.255.1 Constructor & Destructor Documentation****0.15.255.1.1 ~IndexedFunctionInstantiator()** virtual smrat::parser::IndexedFunctionInstantiator::~IndexedFunctionInstantiator () [inline], [virtual]**0.15.255.2 Member Function Documentation****0.15.255.2.1 convert()** template<typename T >  
bool smrat::parser::IndexedFunctionInstantiator::convert (  
  const std::vector< types::TermType > & from,  
  std::vector< T > & to) const [inline]**0.15.255.2.2 operator()()** virtual bool smrat::parser::IndexedFunctionInstantiator::operator() (const std::vector< std::size\_t > &, const std::vector< types::TermType > &, types::TermType &, TheoryError & errors) const [inline], [virtual]**0.15.256 smrat::cadcells::datastructures::IndexedRoot Struct Reference**

Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample).

```
#include <roots.h>
```

**Public Member Functions**

- IndexedRoot (PolyRef p, size\_t i)
- IndexedRoot ()

**Data Fields**

- `PolyRef poly`  
*A multivariate polynomial.*
- `size_t index`  
*The index, must be > 0.*

**0.15.256.1 Detailed Description**

Represents the i-th root of a multivariate polynomial at its main variable (given an appropriate sample).

**0.15.256.2 Constructor & Destructor Documentation**

**0.15.256.2.1 `IndexedRoot()` [1/2]** `smtrat::cadcells::datastructures::IndexedRoot::IndexedRoot ( PolyRef p, size_t i ) [inline]`

**0.15.256.2.2 `IndexedRoot()` [2/2]** `smtrat::cadcells::datastructures::IndexedRoot::IndexedRoot ( ) [inline]`

**0.15.256.3 Field Documentation**

**0.15.256.3.1 `index`** `size_t smtrat::cadcells::datastructures::IndexedRoot::index`  
The index, must be > 0.

**0.15.256.3.2 `poly`** `PolyRef smtrat::cadcells::datastructures::IndexedRoot::poly`  
A multivariate polynomial.

**0.15.257 `smtrat::cadcells::datastructures::IndexedRootOrdering` Class Reference**

Describes an ordering of IndexedRoots.

```
#include <roots.h>
```

**Public Member Functions**

- `void add_leq (RootFunction first, RootFunction second)`
- `void add_less (RootFunction first, RootFunction second)`
- `void add_eq (RootFunction first, RootFunction second)`
- `const auto & data () const`
- `const auto & leq () const`
- `const auto & geq () const`
- `bool holds_transitive (RootFunction first, RootFunction second, bool strict) const`
- `std::optional< RootFunction > holds_transitive (RootFunction first, PolyRef poly, bool strict) const`
- `std::optional< RootFunction > holds_transitive (PolyRef poly, RootFunction second, bool strict) const`
- `void polys (boost::container::flat_set< PolyRef > &result) const`
- `boost::container::flat_set< PolyRef > polys () const`

**0.15.257.1 Detailed Description**

Describes an ordering of IndexedRoots.

## 0.15.257.2 Member Function Documentation

**0.15.257.2.1 add\_eq()** void smtrat::cadcells::datastructures::IndexedRootOrdering::add\_eq ( RootFunction first, RootFunction second ) [inline]

**0.15.257.2.2 add\_leq()** void smtrat::cadcells::datastructures::IndexedRootOrdering::add\_leq ( RootFunction first, RootFunction second ) [inline]

**0.15.257.2.3 add\_less()** void smtrat::cadcells::datastructures::IndexedRootOrdering::add\_less ( RootFunction first, RootFunction second ) [inline]

**0.15.257.2.4 data()** const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::data ( ) const [inline]

**0.15.257.2.5 geq()** const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::geq ( ) const [inline]

**0.15.257.2.6 holds\_transitive() [1/3]** std::optional<RootFunction> smtrat::cadcells::datastructures::IndexedRootOrdering::holds\_transitive ( PolyRef poly, RootFunction second, bool strict ) const [inline]

**0.15.257.2.7 holds\_transitive() [2/3]** std::optional<RootFunction> smtrat::cadcells::datastructures::IndexedRootOrdering::holds\_transitive ( RootFunction first, PolyRef poly, bool strict ) const [inline]

**0.15.257.2.8 holds\_transitive() [3/3]** bool smtrat::cadcells::datastructures::IndexedRootOrdering::holds\_transitive ( RootFunction first, RootFunction second, bool strict ) const [inline]

**0.15.257.2.9 leq()** const auto& smtrat::cadcells::datastructures::IndexedRootOrdering::leq ( ) const [inline]

**0.15.257.2.10 polys()** [1/2] boost::container::flat\_set<PolyRef> smtrat::cadcells::datastructures::IndexedRootOrdering::polys ( ) const [inline]

---

```
0.15.257.2.11 polys() [2/2] void smrat::cadcells::datastructures::IndexedRootOrdering::polys (
 boost::container::flat_set< PolyRef > & result) const [inline]
```

## 0.15.258 smrat::cadcells::datastructures::IndexedRootRelation Struct Reference

A relation between two roots.

```
#include <roots.h>
```

### Data Fields

- RootFunction first
- RootFunction second
- bool is\_strict

#### 0.15.258.1 Detailed Description

A relation between two roots.

#### 0.15.258.2 Field Documentation

**0.15.258.2.1 first** RootFunction smrat::cadcells::datastructures::IndexedRootRelation::first

**0.15.258.2.2 is\_strict** bool smrat::cadcells::datastructures::IndexedRootRelation::is\_strict

**0.15.258.2.3 second** RootFunction smrat::cadcells::datastructures::IndexedRootRelation::second

## 0.15.259 smrat::InequalitiesTable< Settings > Class Template Reference

Datastructure for the [GBModule](#).

```
#include <InequalitiesTable.h>
```

### Public Types

- typedef std::pair< unsigned, Polynomial > CellEntry
- typedef std::tuple< ModuleInput::iterator, carl::Relation, std::list< CellEntry > > RowEntry
- typedef std::map< ModuleInput::const\_iterator, RowEntry, ModuleInput::IteratorCompare > Rows
- typedef std::pair< ModuleInput::const\_iterator, RowEntry > Row
- typedef std::map< carl::Variable, std::pair< TermT, carl::BitVector > > RewriteRules

### Public Member Functions

- InequalitiesTable (GBModule< Settings > \*module)
- Rows::iterator InsertReceivedFormula (ModuleInput::const\_iterator received)
- void pushBacktrackPoint ()
- void popBacktrackPoint (unsigned nrBacktracks)
- Answer reduceWRTGroebnerBasis (const Ideal &gb, const RewriteRules &rules)
- bool reduceWRTGroebnerBasis (typename Rows::iterator, const Ideal &gb, const RewriteRules &rules)
- Answer reduceWRTGroebnerBasis (const std::list< typename Rows::iterator > &ineqToBeReduced, const Ideal &gb, const RewriteRules &rules)
- Answer reduceWRTVariableRewriteRules (const RewriteRules &rules)
- bool reduceWRTVariableRewriteRules (typename Rows::iterator it, const RewriteRules &rules)
- Answer reduceWRTVariableRewriteRules (const std::list< typename Rows::iterator > &ineqToBeReduced, const RewriteRules &rules)
- void removeInequality (ModuleInput::const\_iterator \_formula)
- void print (std::ostream &os=std::cout) const

## Protected Types

- `typedef Settings::PolynomialWithReasons Polynomial`
- `typedef Settings::MultivariateIdeal Ideal`

## Protected Attributes

- `Rows mReducedInequalities`  
*A map of pointers from received iterators to rows.*
- `unsigned mBtnumber`  
*The actual number of backtrackpoints.*
- `GBModule< Settings > * mModule`  
*A pointer to the GBModule which uses this table.*
- `Rows::iterator mNewConstraints`
- `unsigned mLastRestart`

### 0.15.259.1 Detailed Description

```
template<class Settings>
class smrat::InequalitiesTable< Settings >
```

Datastructure for the `GBModule`.

A table of all inequalities and how they are reduced.

Author

Sebastian Junges

### 0.15.259.2 Member Typedef Documentation

**0.15.259.2.1 CellEntry** `template<class Settings >`  
`typedef std::pair<unsigned, Polynomial> smrat::InequalitiesTable< Settings >::CellEntry`

**0.15.259.2.2 Ideal** `template<class Settings >`  
`typedef Settings::MultivariateIdeal smrat::InequalitiesTable< Settings >::Ideal [protected]`

**0.15.259.2.3 Polynomial** `template<class Settings >`  
`typedef Settings::PolynomialWithReasons smrat::InequalitiesTable< Settings >::Polynomial [protected]`

**0.15.259.2.4 RewriteRules** `template<class Settings >`  
`typedef std::map<carl::Variable, std::pair<TermT, carl::BitVector> > smrat::InequalitiesTable< Settings >::RewriteRules`

**0.15.259.2.5 Row** `template<class Settings >`  
`typedef std::pair<ModuleInput::const_iterator, RowEntry> smrat::InequalitiesTable< Settings >::Row`

**0.15.259.2.6 RowEntry** `template<class Settings >`  
`typedef std::tuple<ModuleInput::iterator, carl::Relation, std::list<CellEntry> > smrat::InequalitiesTable< Settings >::RowEntry`

**0.15.259.2.7 Rows** template<class Settings >  
typedef std::map<ModuleInput::const\_iterator, RowEntry, ModuleInput::IteratorCompare> smrat::InequalitiesTable

### 0.15.259.3 Constructor & Destructor Documentation

**0.15.259.3.1 InequalitiesTable()** template<class Settings >  
smrat::InequalitiesTable< Settings >::InequalitiesTable (   
    GBModule< Settings > \* module )

### 0.15.259.4 Member Function Documentation

**0.15.259.4.1 InsertReceivedFormula()** template<class Settings >  
Rows::iterator smrat::InequalitiesTable< Settings >::InsertReceivedFormula (   
    ModuleInput::const\_iterator received )

**0.15.259.4.2 popBacktrackPoint()** template<class Settings >  
void smrat::InequalitiesTable< Settings >::popBacktrackPoint (   
    unsigned nrBacktracks )

**0.15.259.4.3 print()** template<class Settings >  
void smrat::InequalitiesTable< Settings >::print (   
    std::ostream & os = std::cout ) const

**0.15.259.4.4 pushBacktrackPoint()** template<class Settings >  
void smrat::InequalitiesTable< Settings >::pushBacktrackPoint ( )

**0.15.259.4.5 reduceWRTGroebnerBasis() [1/3]** template<class Settings >  
Answer smrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (   
    const Ideal & gb,  
    const RewriteRules & rules )

**0.15.259.4.6 reduceWRTGroebnerBasis() [2/3]** template<class Settings >  
Answer smrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (   
    const std::list< typename Rows::iterator > & ineqToBeReduced,  
    const Ideal & gb,  
    const RewriteRules & rules )

**0.15.259.4.7 reduceWRTGroebnerBasis() [3/3]** template<class Settings >  
bool smrat::InequalitiesTable< Settings >::reduceWRTGroebnerBasis (   
    typename Rows::iterator ,  
    const Ideal & gb,  
    const RewriteRules & rules )

**0.15.259.4.8 reduceWRTVariableRewriteRules()** [1/3] template<class Settings >  
Answer smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules ( const RewriteRules & rules )

**0.15.259.4.9 reduceWRTVariableRewriteRules()** [2/3] template<class Settings >  
Answer smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules ( const std::list< typename Rows::iterator > & ineqToBeReduced, const RewriteRules & rules )

**0.15.259.4.10 reduceWRTVariableRewriteRules()** [3/3] template<class Settings > bool smtrat::InequalitiesTable< Settings >::reduceWRTVariableRewriteRules ( typename Rows::iterator it, const RewriteRules & rules )

**0.15.259.4.11 removeInequality()** template<class Settings > void smtrat::InequalitiesTable< Settings >::removeInequality ( ModuleInput::const\_iterator \_formula )

## 0.15.259.5 Field Documentation

**0.15.259.5.1 mBtnumber** template<class Settings > unsigned smtrat::InequalitiesTable< Settings >::mBtnumber [protected]  
The actual number of backtrackpoints.

**0.15.259.5.2 mLastRestart** template<class Settings > unsigned smtrat::InequalitiesTable< Settings >::mLastRestart [protected]

**0.15.259.5.3 mModule** template<class Settings > GBModule<Settings>\* smtrat::InequalitiesTable< Settings >::mModule [protected]  
A pointer to the [GBModule](#) which uses this table.

**0.15.259.5.4 mNewConstraints** template<class Settings > Rows::iterator smtrat::InequalitiesTable< Settings >::mNewConstraints [protected]

**0.15.259.5.5 mReducedInequalities** template<class Settings > Rows smtrat::InequalitiesTable< Settings >::mReducedInequalities [protected]  
A map of pointers from received iterators to rows.

## 0.15.260 smtrat::lra::Bound< T1, T2 >::Info Struct Reference

Stores some additional information for a bound.  
`#include <Bound.h>`

### Public Member Functions

- [Info \(ModuleInput::iterator \\_position\)](#)

## Data Fields

- int **updated**  
*A value to store the information whether this bounds constraint has already been processed (=0), must be processed (>0) or must be removed from consideration (<0).*
- **ModuleInput::iterator position**  
*The position of this bounds constraint in a formula, if it has been processed already.*
- **FormulaT neqRepresentation**  
*If this bound corresponds to a constraint being  $p < 0$  or  $p > 0$  for a polynomial  $p$ , we store here the constraint  $p \neq 0$ .*
- bool **exists**  
*A flag which is only false, if this bound has been created for a constraint having != as relation symbol, i.e.*
- const **Bound< T1, T2 > \* complement**

### 0.15.260.1 Detailed Description

```
template<typename T1, typename T2>
struct smrat::lra::Bound< T1, T2 >::Info
```

Stores some additional information for a bound.

### 0.15.260.2 Constructor & Destructor Documentation

```
0.15.260.2.1 Info() template<typename T1 , typename T2 >
smrat::lra::Bound< T1, T2 >::Info::Info (
 ModuleInput::iterator _position) [inline]
```

### 0.15.260.3 Field Documentation

```
0.15.260.3.1 complement template<typename T1 , typename T2 >
const Bound<T1, T2>* smrat::lra::Bound< T1, T2 >::Info::complement
```

```
0.15.260.3.2 exists template<typename T1 , typename T2 >
bool smrat::lra::Bound< T1, T2 >::Info::exists
A flag which is only false, if this bound has been created for a constraint having != as relation symbol, i.e.
p!=0, and not yet for the constraint p<0 or p>0.
```

```
0.15.260.3.3 neqRepresentation template<typename T1 , typename T2 >
FormulaT smrat::lra::Bound< T1, T2 >::Info::neqRepresentation
If this bound corresponds to a constraint being $p < 0$ or $p > 0$ for a polynomial p , we store here the constraint $p \neq 0$.
```

```
0.15.260.3.4 position template<typename T1 , typename T2 >
ModuleInput::iterator smrat::lra::Bound< T1, T2 >::Info::position
The position of this bounds constraint in a formula, if it has been processed already.
Otherwise it points to the end of a formula.
```

```
0.15.260.3.5 updated template<typename T1 , typename T2 >
int smrat::lra::Bound< T1, T2 >::Info::updated
A value to store the information whether this bounds constraint has already been processed (=0), must be processed
(>0) or must be removed from consideration (<0).
```

## 0.15.261 smtrat::mcsat::InformationGetter Struct Reference

```
#include <MCASATMixin.h>
```

### Data Fields

- std::function< Minisat::lbool(Minisat::Var)> getVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getLitValue
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< int(Minisat::Var)> getDecisionLevel
- std::function< int(Minisat::Var)> getTrailIndex
- std::function< Minisat::CRef(Minisat::Var)> getReason
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< const Minisat::vec< Minisat::CRef > &()> getClauses
- std::function< const Minisat::vec< Minisat::CRef > &()> getLearntClauses
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< bool(const FormulaT &)> isAbstractedFormula
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::vec< Minisat::Watcher > &(Minisat::Lit)> getWatches
- std::function< Minisat::Var()> newVar

### 0.15.261.1 Field Documentation

**0.15.261.1.1 `abstractVariable`** std::function<Minisat::Var(const FormulaT&)> smtrat::mcsat::InformationGetter::abstractVariable

**0.15.261.1.2 `getBoolVarValue`** std::function<Minisat::lbool(Minisat::Var)> smtrat::mcsat::InformationGetter::getBoolVarValue

**0.15.261.1.3 `getClause`** std::function<const Minisat::Clause &(Minisat::CRef)> smtrat::mcsat::InformationGetter::getClause

**0.15.261.1.4 `getClauses`** std::function<const Minisat::vec<Minisat::CRef>&()> smtrat::mcsat::InformationGetter::getClauses

**0.15.261.1.5 `getDecisionLevel`** std::function<int(Minisat::Var)> smtrat::mcsat::InformationGetter::getDecisionLevel

**0.15.261.1.6 `getLearntClauses`** std::function<const Minisat::vec<Minisat::CRef>&()> smtrat::mcsat::InformationGetter::getLearntClauses

**0.15.261.1.7 `getLitValue`** std::function<Minisat::lbool(Minisat::Lit)> smtrat::mcsat::InformationGetter::getLitValue

**0.15.261.1.8 getReason** std::function<Minisat::CRef(Minisat::Var)> smtrat::mcsat::Information<~> Getter::getReason

**0.15.261.1.9 getTrailIndex** std::function<int (Minisat::Var)> smtrat::mcsat::Information<~> Getter::getTrailIndex

**0.15.261.1.10 getVarValue** std::function<Minisat::lbool (Minisat::Var)> smtrat::mcsat::Information<~> Getter::getVarValue

**0.15.261.1.11 getWatches** std::function<const Minisat::vec<Minisat::Watcher>& (Minisat::Lit)> smtrat::mcsat::InformationGetter::getWatches

**0.15.261.1.12 isAbstractedFormula** std::function<bool (const FormulaT&)> smtrat::mcsat::Information<~> Getter::isAbstractedFormula

**0.15.261.1.13 isTheoryAbstraction** std::function<bool (Minisat::Var)> smtrat::mcsat::Information<~> Getter::isTheoryAbstraction

**0.15.261.1.14 newVar** std::function<Minisat::Var ()> smtrat::mcsat::InformationGetter::newVar

**0.15.261.1.15 reabstractLiteral** std::function<const FormulaT& (Minisat::Lit)> smtrat::mcsat::Information<~> InformationGetter::reabstractLiteral

**0.15.261.1.16 reabstractVariable** std::function<const FormulaT& (Minisat::Var)> smtrat::mcsat::Information<~> InformationGetter::reabstractVariable

## 0.15.262 smtrat::parser::Instantiator< V, T > Struct Template Reference

```
#include <FunctionInstantiator.h>
```

### Public Member Functions

- template<typename Res >  
  bool **operator()** (const Res &)
- bool **operator()** (carl::Variable v)
- template<typename TYPE = T>  
  std::enable\_if< std::is\_same< TYPE, Poly >::value, bool >::type **operator()** (const Poly &p)
- bool **operator()** (const FormulaT &f)
- template<typename VAR = V>  
  std::enable\_if< std::is\_same< VAR, types::BVVariable >::value, bool >::type **operator()** (const types::BVVariable &v)
- template<typename VAR = V>  
  std::enable\_if< std::is\_same< VAR, types::BVVariable >::value, bool >::type **operator()** (const types::BVTerm &t)
- bool **instantiate** (V v, const T &repl, types::TermType &subject)

## Protected Attributes

- V var
- T replacement
- types::TermType result

### 0.15.262.1 Member Function Documentation

**0.15.262.1.1 instantiate()** template<typename V , typename T >  
bool smtrat::parser::Instantiator< V, T >::instantiate (  
 V v,  
 const T & repl,  
 types::TermType & subject ) [inline]

**0.15.262.1.2 operator() [1/6]** template<typename V , typename T >  
bool smtrat::parser::Instantiator< V, T >::operator() (  
 carl::Variable v ) [inline]

**0.15.262.1.3 operator() [2/6]** template<typename V , typename T >  
bool smtrat::parser::Instantiator< V, T >::operator() (  
 const FormulaT & f ) [inline]

**0.15.262.1.4 operator() [3/6]** template<typename V , typename T >  
template<typename TYPE = T>  
std::enable\_if<std::is\_same<TYPE,Poly>::value, bool>::type smtrat::parser::Instantiator< V,  
T >::operator() (  
 const Poly & p ) [inline]

**0.15.262.1.5 operator() [4/6]** template<typename V , typename T >  
template<typename Res >  
bool smtrat::parser::Instantiator< V, T >::operator() (  
 const Res & ) [inline]

**0.15.262.1.6 operator() [5/6]** template<typename V , typename T >  
template<typename VAR = V>  
std::enable\_if<std::is\_same<VAR,types::BVVariable>::value, bool>::type smtrat::parser::Instantiator<  
V, T >::operator() (  
 const types::BVTerm & t ) [inline]

**0.15.262.1.7 operator() [6/6]** template<typename V , typename T >  
template<typename VAR = V>  
std::enable\_if<std::is\_same<VAR,types::BVVariable>::value, bool>::type smtrat::parser::Instantiator<  
V, T >::operator() (  
 const types::BVVariable & v ) [inline]

### 0.15.262.2 Field Documentation

**0.15.262.2.1 replacement** template<typename V , typename T >  
T smtrat::parser::Instantiator< V, T >::replacement [protected]

**0.15.262.2.2 result** template<typename V , typename T >  
types::TermType smtrat::parser::Instantiator< V, T >::result [protected]

**0.15.262.2.3 var** template<typename V , typename T >  
V smtrat::parser::Instantiator< V, T >::var [protected]

## 0.15.263 smtrat::parser::InstructionHandler Class Reference

```
#include <InstructionHandler.h>
```

### Public Types

- **typedef types::AttributeValue Value**

### Public Member Functions

- void **addInstruction** (std::function< void()> bind)
- bool **hasInstructions** () const
- void **runInstructions** ()
- void **setArtificialVariables** (std::vector< smtrat::ModelVariable > &&vars)
- void **cleanModel** (smtrat::Model &model) const
- bool **has\_info** (const std::string &key) const
- const auto & **get\_info** (const std::string &key) const
- template<typename T >  
T **option** (const std::string &key) const
- bool **printInstruction** () const
- **InstructionHandler** ()
- virtual ~**InstructionHandler** ()
- std::ostream & **diagnostic** ()
- void **diagnostic** (const std::string &s)
- std::ostream & **regular** ()
- void **regular** (const std::string &s)
- **OutputWrapper error** ()
- **OutputWrapper warn** ()
- **OutputWrapper info** ()
- virtual void **add** (const **FormulaT** &f)=0
- virtual void **addSoft** (const **FormulaT** &f, Rational weight, const std::string &id)=0
- virtual void **annotateName** (const **FormulaT** &f, const std::string &name)=0
- virtual void **check** ()=0
- virtual void **declareFun** (const carl::Variable &)=0
- virtual void **declareSort** (const std::string &, const unsigned &)=0
- virtual void **defineSort** (const std::string &, const std::vector< std::string > &, const carl::Sort &)=0
- virtual void **echo** (const std::string &s)
- virtual void **eliminateQuantifiers** (const qe::QEQuery &q)=0
- virtual void **exit** ()=0
- virtual void **getAllModels** ()=0
- virtual void **getAssertions** ()=0
- virtual void **getAssignment** ()=0
- void **getInfo** (const std::string &key)
- virtual void **getModel** ()=0
- virtual void **getObjectives** ()=0

- void `getOption` (const std::string &key)
- virtual void `getProof` ()=0
- virtual void `getUnsatCore` ()=0
- virtual void `getValue` (const std::vector< carl::Variable > &)=0
- virtual void `addObjective` (const Poly &p, OptimizationType ot)=0
- virtual void `pop` (std::size\_t)=0
- virtual void `push` (std::size\_t)=0
- virtual void `reset` ()
- virtual void `resetAssertions` ()=0
- void `setInfo` (const Attribute &attr)
- virtual void `setLogic` (const carl::Logic &)=0
- void `setOption` (const Attribute &option)

### Protected Member Functions

- void `setStream` (const std::string &s, std::ostream &os)

### Protected Attributes

- VariantMap< std::string, Value > infos
- VariantMap< std::string, Value > options
- std::ostream `mRegular`
- std::ostream `mDiagnostic`
- std::map< std::string, std::ofstream > streams

## 0.15.263.1 Member Typedef Documentation

**0.15.263.1.1 Value** `typedef types::AttributeValue smrat::parser::InstructionHandler::Value`

## 0.15.263.2 Constructor & Destructor Documentation

**0.15.263.2.1 InstructionHandler()** `smrat::parser::InstructionHandler::InstructionHandler ( ) [inline]`

**0.15.263.2.2 ~InstructionHandler()** `virtual smrat::parser::InstructionHandler::~InstructionHandler ( ) [inline], [virtual]`

## 0.15.263.3 Member Function Documentation

**0.15.263.3.1 add()** `virtual void smrat::parser::InstructionHandler::add ( const FormulaT & f ) [pure virtual]`

Implemented in `smrat::parseformula::FormulaCollector`, and `smrat::Executor< Strategy >`.

**0.15.263.3.2 addInstruction()** `void smrat::parser::InstructionHandler::addInstruction ( std::function< void()> bind ) [inline]`

**0.15.263.3.3 `addObjective()`** `virtual void smtrat::parser::InstructionHandler::addObjective ( const Poly & p, OptimizationType ot ) [pure virtual]`

Implemented in [smtrat::Executor< Strategy >](#), and [smtrat::parseformula::FormulaCollector](#).

**0.15.263.3.4 `addSoft()`** `virtual void smtrat::parser::InstructionHandler::addSoft ( const FormulaT & f, Rational weight, const std::string & id ) [pure virtual]`

Implemented in [smtrat::Executor< Strategy >](#), and [smtrat::parseformula::FormulaCollector](#).

**0.15.263.3.5 `annotateName()`** `virtual void smtrat::parser::InstructionHandler::annotateName ( const FormulaT & f, const std::string & name ) [pure virtual]`

Implemented in [smtrat::Executor< Strategy >](#), and [smtrat::parseformula::FormulaCollector](#).

**0.15.263.3.6 `check()`** `virtual void smtrat::parser::InstructionHandler::check () [pure virtual]`

Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.7 `cleanModel()`** `void smtrat::parser::InstructionHandler::cleanModel ( smtrat::Model & model ) const [inline]`

**0.15.263.3.8 `declareFun()`** `virtual void smtrat::parser::InstructionHandler::declareFun ( const carl::Variable & ) [pure virtual]`

Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.9 `declareSort()`** `virtual void smtrat::parser::InstructionHandler::declareSort ( const std::string & , const unsigned & ) [pure virtual]`

Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.10 `defineSort()`** `virtual void smtrat::parser::InstructionHandler::defineSort ( const std::string & , const std::vector< std::string > & , const carl::Sort & ) [pure virtual]`

Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.11 `diagnostic() [1/2]`** `std::ostream& smtrat::parser::InstructionHandler::diagnostic ( ) [inline]`

**0.15.263.3.12 `diagnostic() [2/2]`** `void smtrat::parser::InstructionHandler::diagnostic ( const std::string & s ) [inline]`

**0.15.263.3.13 `echo()`** `virtual void smtrat::parser::InstructionHandler::echo ( const std::string & s ) [inline], [virtual]`

**0.15.263.3.14 `eliminateQuantifiers()`** virtual void smtrat::parser::InstructionHandler::eliminate←  
Quantifiers ( const QEQuery & q ) [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.15 `error()`** OutputWrapper smtrat::parser::InstructionHandler::error () [inline]

**0.15.263.3.16 `exit()`** virtual void smtrat::parser::InstructionHandler::exit () [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.17 `get_info()`** const auto& smtrat::parser::InstructionHandler::get\_info ( const std::string & key ) const [inline]

**0.15.263.3.18 `getAllModels()`** virtual void smtrat::parser::InstructionHandler::getAllModels () [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.19 `getAssertions()`** virtual void smtrat::parser::InstructionHandler::getAssertions () [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.20 `getAssignment()`** virtual void smtrat::parser::InstructionHandler::getAssignment () [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.21  `getInfo()`** void smtrat::parser::InstructionHandler::getInfo ( const std::string & key ) [inline]

**0.15.263.3.22 `getModel()`** virtual void smtrat::parser::InstructionHandler::getModel () [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.23 `getObjectives()`** virtual void smtrat::parser::InstructionHandler::getObjectives () [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.24 `getOption()`** void smtrat::parser::InstructionHandler::getOption ( const std::string & key ) [inline]

**0.15.263.3.25 `getProof()`** virtual void smtrat::parser::InstructionHandler::getProof () [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.26 `getUnsatCore()`** `virtual void smtrat::parser::InstructionHandler::getUnsatCore ( )`  
[pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.27 `getValue()`** `virtual void smtrat::parser::InstructionHandler::getValue (`  
`const std::vector< carl::Variable > & )` [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.28 `has_info()`** `bool smtrat::parser::InstructionHandler::has_info (`  
`const std::string & key ) const` [inline]

**0.15.263.3.29 `hasInstructions()`** `bool smtrat::parser::InstructionHandler::hasInstructions ( )`  
const [inline]

**0.15.263.3.30 `info()`** `OutputWrapper smtrat::parser::InstructionHandler::info ( )` [inline]

**0.15.263.3.31 `option()`** `template<typename T >`  
`T smtrat::parser::InstructionHandler::option (`  
`const std::string & key ) const` [inline]

**0.15.263.3.32 `pop()`** `virtual void smtrat::parser::InstructionHandler::pop (`  
`std::size_t )` [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.33 `printInstruction()`** `bool smtrat::parser::InstructionHandler::printInstruction ( )`  
const [inline]

**0.15.263.3.34 `push()`** `virtual void smtrat::parser::InstructionHandler::push (`  
`std::size_t )` [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.35 `regular()` [1/2]** `std::ostream& smtrat::parser::InstructionHandler::regular ( )`  
[inline]

**0.15.263.3.36 `regular()` [2/2]** `void smtrat::parser::InstructionHandler::regular (`  
`const std::string & s )` [inline]

**0.15.263.3.37 `reset()`** `virtual void smtrat::parser::InstructionHandler::reset ( )` [inline],  
[virtual]  
Reimplemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.38 `resetAssertions()`** `virtual void smtrat::parser::InstructionHandler::resetAssertions( )` [pure virtual]  
Implemented in [smtrat::parseformula::FormulaCollector](#), and [smtrat::Executor< Strategy >](#).

**0.15.263.3.39 `runInstructions()`** `void smtrat::parser::InstructionHandler::runInstructions( )` [inline]

**0.15.263.3.40 `setArtificialVariables()`** `void smtrat::parser::InstructionHandler::setArtificialVariables( std::vector< smtrat::ModelVariable > && vars )` [inline]

**0.15.263.3.41 `setInfo()`** `void smtrat::parser::InstructionHandler::setInfo( const Attribute & attr )` [inline]

**0.15.263.3.42 `setLogic()`** `virtual void smtrat::parser::InstructionHandler::setLogic( const carl::Logic & )` [pure virtual]

Implemented in [smtrat::Executor< Strategy >](#), and [smtrat::parseformula::FormulaCollector](#).

**0.15.263.3.43 `setOption()`** `void smtrat::parser::InstructionHandler::setOption( const Attribute & option )` [inline]

**0.15.263.3.44 `setStream()`** `void smtrat::parser::InstructionHandler::setStream( const std::string & s, std::ostream & os )` [inline], [protected]

**0.15.263.3.45 `warn()`** `OutputWrapper smtrat::parser::InstructionHandler::warn( )` [inline]

## 0.15.263.4 Field Documentation

**0.15.263.4.1 `infos`** `VariantMap<std::string, Value> smtrat::parser::InstructionHandler::infos` [protected]

**0.15.263.4.2 `mDiagnostic`** `std::ostream smtrat::parser::InstructionHandler::mDiagnostic` [protected]

**0.15.263.4.3 `mRegular`** `std::ostream smtrat::parser::InstructionHandler::mRegular` [protected]

**0.15.263.4.4 `options`** `VariantMap<std::string, Value> smtrat::parser::InstructionHandler::options` [protected]

**0.15.263.4.5 `streams`** `std::map<std::string, std::ofstream> smtrat::parser::InstructionHandler::streams` [protected]

## 0.15.264 Minisat::Int64Range Struct Reference

```
#include <Options.h>
```

### Public Member Functions

- `Int64Range (int64_t b, int64_t e)`

### Data Fields

- `int64_t begin`
- `int64_t end`

#### 0.15.264.1 Constructor & Destructor Documentation

**0.15.264.1.1 `Int64Range()`** `Minisat::Int64Range::Int64Range (`  
 `int64_t b,`  
 `int64_t e ) [inline]`

#### 0.15.264.2 Field Documentation

**0.15.264.2.1 `begin`** `int64_t Minisat::Int64Range::begin`

**0.15.264.2.2 `end`** `int64_t Minisat::Int64Range::end`

## 0.15.265 smtrat::IntBlastModule< Settings > Class Template Reference

```
#include <IntBlastModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
- `IntBlastModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~IntBlastModule ()`
- `bool informCore (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `bool addCore (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `void removeCore (ModuleInput::const_iterator _subformula)`  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- `void updateModel () const`  
*Updates the current assignment into the model.*
- `Answer checkCore ()`  
*Checks the received formula for consistency.*
- `bool inform (const FormulaT &_constraint)`

- **Informs the module about the given constraint.**
- void **deinform** (const **FormulaT** &\_constraint)
 

*The inverse of informing about a constraint.*
- bool **add** (**ModuleInput::const\_iterator** \_subformula)
 

*The module has to take the given sub-formula of the received formula into account.*
- virtual **Answer check** (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)
 

*Checks the received formula for consistency.*
- virtual void **remove** (**ModuleInput::const\_iterator** \_subformula)
 

*Removes everything related to the given sub-formula of the received formula.*
- virtual void **updateAllModels** ()
 

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const
 

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned **currentlySatisfiedByBackend** (const **FormulaT** &\_formula) const
- virtual unsigned **currentlySatisfied** (const **FormulaT** &) const
- bool **receivedVariable** (carl::Variable::Arg \_var) const
- **Answer solverState** () const
- std::size\_t **id** () const
- void **setId** (std::size\_t \_id)
 

*Sets this modules unique ID to identify itself.*
- **thread\_priority threadPriority** () const
- void **setThreadPriority** (**thread\_priority** \_threadPriority)
 

*Sets the priority of this module to get a thread for running its check procedure.*
- const **ModuleInput** \* **pReceivedFormula** () const
- const **ModuleInput** & **rReceivedFormula** () const
- const **ModuleInput** \* **pPassedFormula** () const
- const **ModuleInput** & **rPassedFormula** () const
- const **Model** & **model** () const
- const std::vector< **Model** > & **allModels** () const
- const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
- const std::vector< **Module** \* > & **usedBackends** () const
- const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
- const carl::FastSet< **FormulaT** > & **informedConstraints** () const
- void **addLemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lt=LemmaType::NORMAL, const **FormulaT** &\_preferredFormula=**FormulaT**(carl::FormulaType::TRUE))
 

*Stores a lemma being a valid formula.*
- bool **hasLemmas** ()
 

*Checks whether this module or any of its backends provides any lemmas.*
- void **clearLemmas** ()
 

*Deletes all yet found lemmas.*
- const std::vector< **Lemma** > & **lemmas** () const
- **ModuleInput::const\_iterator** **firstUncheckedReceivedSubformula** () const
- **ModuleInput::const\_iterator** **firstSubformulaToPass** () const
- void **receivedFormulaChecked** ()
 

*Notifies that the received formulas has been checked.*
- const **smrat::Conditionals** & **answerFound** () const
- bool **isPreprocessor** () const
- carl::Variable **objective** () const
- bool **is\_minimizing** () const
- void **excludeNotReceivedVariablesFromModel** () const
 

*Excludes all variables from the current model, which do not occur in the received formula.*
- void **updateLemmas** ()

- Stores all lemmas of any backend of this module in its own lemma vector.
- void `collectTheoryPropagations ()`
  - Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`
  - Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `checkInSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _insubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`
  - Prints everything relevant of the solver.*
- void `print (const std::string &_initiation="***") const`
  - Prints the vector of the received formula.*
- void `printReceivedFormula (const std::string &_initiation="***") const`
  - Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`
  - Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`
  - Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`
  - Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5
  - The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))
  - Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0
  - The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`
  - Reusable splitting variables.*

### Protected Member Functions

- virtual void `deinformCore (const FormulaT &_constraint)`
  - The inverse of informing about a constraint.*
- bool `anAnswerFound () const`
  - Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel () const`
  - Clears the assignment, if any was found.*
- void `clearModels () const`
  - Clears all assignments, if any was found.*
- void `cleanModel () const`

- Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin ()`
  - `ModuleInput::iterator passedFormulaEnd ()`
  - `void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
  - `const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`

*Gets the origins of the passed formula at the given position.*
  - `void getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
  - `void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
  - `std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
  - `std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
  - `void informBackends (const FormulaT &_constraint)`

*Informs all backends of this module about the given constraint.*
  - `virtual void addConstraintToInform (const FormulaT &_constraint)`

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
  - `std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`

*Copies the given sub-formula of the received formula to the passed formula.*
  - `bool originInReceivedFormula (const FormulaT &_origin) const`
  - `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)`

*Adds the given formula to the passed formula with no origin.*
  - `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`

*Adds the given formula to the passed formula.*
  - `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`

*Adds the given formula to the passed formula.*
  - `void generateTrivialInfeasibleSubset ()`

*Stores the trivial infeasible subset being the set of received formulas.*
  - `void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

*Stores an infeasible subset consisting only of the given received formula.*
  - `std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const`
  - `void getInfeasibleSubsets ()`

*Copies the infeasible subsets of the passed formula.*
  - `std::vector< FormulaSetT > getInfeasibleSubsets (const Model & backend) const`

*Get the infeasible subsets the given backend provides.*
  - `const Model & backendsModel () const`

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - `void getBackendsModel () const`
  - `void getBackendsAllModels () const`

*Stores all models of a backend in the list of all models of this module.*
  - `virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*
  - `virtual Answer runBackends ()`
  - `virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
  - `void clearPassedFormula ()`
  - `std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`

*Merges the two vectors of sets into the first one.*

- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename `Poly::PolyType` &`_branchingPolynomial`, const `Rational` &`_branchingValue`) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const `Poly` &`_polynomial`, bool `_integral`, const `Rational` &`_value`, std::vector< `FormulaT` > &&`_premise`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (const `Poly` &`_polynomial`, bool `_integral`, const `Rational` &`_value`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`, const std::vector< `FormulaT` > &`_premise=std::vector< FormulaT >()`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (carl::Variable::Arg `_var`, const `Rational` &`_value`, std::vector< `FormulaT` > &&`_premise`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (carl::Variable::Arg `_var`, const `Rational` &`_value`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`, const std::vector< `FormulaT` > &`_premise=std::vector< FormulaT >()`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint `p!=0`.*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &`formula`)
 

*Adds a formula to the `InformationRelevantFormula`.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
 

*Gets all `InformationRelevantFormulas`.*
- bool `isLemmaLevel` (`LemmaLevel` `level`)
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &`_modelA`, const `Model` &`_modelB`)
 

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- `std::vector< Module * > mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- `std::vector< Module * > mAllBackends`  
*The backends of this module which have been used.*
- `std::vector< Lemma > mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- `carl::FastSet< FormulaT > mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.265.1 Member TypeDef Documentation

**0.15.265.1.1 SettingsType** `template<class Settings >`  
`typedef Settings smtrat::IntBlastModule< Settings >::SettingsType`

### 0.15.265.2 Member Enumeration Documentation

**0.15.265.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.265.3 Constructor & Destructor Documentation

**0.15.265.3.1 IntBlastModule()** `template<class Settings >`  
`smtrat::IntBlastModule< Settings >::IntBlastModule (`  
 `const ModuleInput * _formula,`  
 `Conditionals & _conditionals,`  
 `Manager * _manager = NULL )`

```
0.15.265.3.2 ~IntBlastModule() template<class Settings >
smrat::IntBlastModule< Settings >::~IntBlastModule ()
```

#### 0.15.265.4 Member Function Documentation

**0.15.265.4.1 add()** bool smrat::Module::add (   
`ModuleInput::const_iterator _subformula` ) [inherited]

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.265.4.2 addConstraintToInform()** void smrat::Module::addConstraintToInform (   
`const FormulaT & _constraint` ) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

##### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smrat::SATModule< Settings >](#).

**0.15.265.4.3 addCore()** template<class Settings >
bool smrat::IntBlastModule< Settings >::addCore (   
`ModuleInput::const_iterator _subformula` ) [virtual]

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.265.4.4 addInformationRelevantFormula()** void smrat::Module::addInformationRelevantFormula (   
`const FormulaT & formula` ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

##### Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

---

**0.15.265.4.5 addLemma()** void smtrat::Module::addLemma (

```
const FormulaT & _lemma,
const LemmaType & _lt = LemmaType::NORMAL,
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.265.4.6 addOrigin()** void smtrat::Module::addOrigin (

```
ModuleInput::iterator _formula,
const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

**0.15.265.4.7 addReceivedSubformulaToPassedFormula()** std::pair<ModuleInput::iterator,bool>  
smtrat::Module::addReceivedSubformulaToPassedFormula (

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>_subformula</i> | The sub-formula of the received formula to copy. |
|--------------------|--------------------------------------------------|

**0.15.265.4.8 addSubformulaToPassedFormula()** [1/3] std::pair<ModuleInput::iterator,bool> smtrat->  
::Module::addSubformulaToPassedFormula (

```
const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula. |
|-----------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.265.4.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smrat->
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                 |
| <i>_origin</i>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.265.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.265.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline],
[inherited]
```

**Returns**

All satisfying assignments, if existent.

**0.15.265.4.12 `anAnswerFound()`** `bool smrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.265.4.13 `answerFound()`** `const smrat::Conditionals& smrat::Module::answerFound ( ) const [inline], [inherited]`

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.265.4.14 `backendsModel()`** `const Model & smrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.265.4.15 `branchAt() [1/4]`** `bool smrat::Module::branchAt ( carl::Variable::Arg _var, const Rational & _value, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]`

**0.15.265.4.16 `branchAt() [2/4]`** `bool smrat::Module::branchAt ( carl::Variable::Arg _var, const Rational & _value, std::vector< FormulaT > && _premise, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]`

**0.15.265.4.17 `branchAt() [3/4]`** `bool smrat::Module::branchAt ( const Poly & _polynomial, bool _integral, const Rational & _value, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]`

**0.15.265.4.18 `branchAt() [4/4]`** `bool smrat::Module::branchAt ( const Poly & _polynomial, bool _integral, const Rational & _value,`

```
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

#### 0.15.265.4.19 `check()`

```
Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

#### 0.15.265.4.20 `checkCore()`

```
template<class Settings >
Answer smtrat::IntBlastModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.265.4.21 checkInfSubsetForMinimality() void smrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infsSubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.265.4.22 checkModel() unsigned smrat::Module::checkModel () const [protected], [inherited]
```

#### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

```
0.15.265.4.23 cleanModel() void smrat::Module::cleanModel () const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

```
0.15.265.4.24 clearLemmas() void smrat::Module::clearLemmas () [inline], [inherited]
```

Deletes all yet found lemmas.

```
0.15.265.4.25 clearModel() void smrat::Module::clearModel () const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

```
0.15.265.4.26 clearModels() void smrat::Module::clearModels () const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

```
0.15.265.4.27 clearPassedFormula() void smrat::Module::clearPassedFormula () [protected], [inherited]
```

```
0.15.265.4.28 collectOrigins() [1/2] void smrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

```
0.15.265.4.29 collectOrigins() [2/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                 |                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <i>_origins</i> | The set in which to store the origins.                                                               |

```
0.15.265.4.30 collectTheoryPropagations() void smtrat::Module::collectTheoryPropagations () [inherited]
```

```
0.15.265.4.31 constraintsToInform() const carl::FastSet<FormulaT>& smtrat::Module::constraints←ToInform () const [inline], [inherited]
```

#### Returns

The constraints which the backends must be informed about.

```
0.15.265.4.32 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

```
0.15.265.4.33 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.265.4.34 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                    |                                                         |
|--------------------|---------------------------------------------------------|
| <i>_constraint</i> | The constraint to remove from internal data structures. |
|--------------------|---------------------------------------------------------|

---

**0.15.265.4.35 `deinformCore()`** `virtual void smtrat::Module::deinformCore (`  
`const FormulaT & _constraint )` [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.265.4.36 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (`  
`const std::vector< FormulaT > & origins ) const` [protected], [inherited]

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.265.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`  
`ModuleInput::iterator _subformula,`  
`bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
 Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.265.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.265.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`  
`const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

## Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

## Returns

**0.15.265.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.265.4.41 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

## Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.265.4.42 `freeSplittingVariable()`** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.265.4.43 `generateTrivialInfeasibleSubset()`** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.265.4.44 `getBackendsAllModels()`** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.265.4.45 `getBackendsModel()`** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.265.4.46 `getInfeasibleSubsets() [1/2]`** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.265.4.47 `getInfeasibleSubsets() [2/2]`** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.265.4.48 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.265.4.49 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.265.4.50 `getOrigins()` [1/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.265.4.51 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.265.4.52 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.265.4.53 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified( ) [virtual], [inherited]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.265.4.54 `hasLemmas()`** `bool smtrat::Module::hasLemmas( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

**0.15.265.4.55 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.265.4.56 `id()`** `std::size_t smtrat::Module::id( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.265.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.265.4.58 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.265.4.59 informBackends()** `void smrat::Module::informBackends (`

`const FormulaT & _constraint ) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.265.4.60 informCore()** `template<class Settings >`

`bool smrat::IntBlastModule< Settings >::informCore (`

`const FormulaT & _constraint ) [virtual]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.265.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informed→`

`Constraints ( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.265.4.62 init()** `template<class Settings >`

`void smrat::IntBlastModule< Settings >::init ( ) [virtual]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not and will not be called more than once and only before the first runBackends call.

Reimplemented from [smrat::Module](#).

**0.15.265.4.63 is\_minimizing()** `bool smrat::Module::is_minimizing ( ) const [inline], [inherited]`**0.15.265.4.64 isLemmaLevel()** `bool smrat::Module::isLemmaLevel (`

`LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.265.4.65 `isPreprocessor()`** `bool smtrat::Module::isPreprocessor () const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.265.4.66 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas () const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.265.4.67 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.265.4.68 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.265.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                |                                |
|----------------|--------------------------------|
| <i>_modelA</i> | The first model to check for.  |
| <i>_modelB</i> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.265.4.70 `moduleName()`** `template<class Settings >`  
`std::string smtrat::IntBlastModule< Settings >::moduleName ( ) const [inline], [virtual]`

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.265.4.71 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.265.4.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`  
`const FormulaT & _origin ) const [protected], [inherited]`

**0.15.265.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin (`  
`) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.265.4.74 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.265.4.75 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.265.4.76 `pReceivedFormula()`** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.265.4.77 `print()`** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.265.4.78 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.265.4.79 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.265.4.80 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.265.4.81 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.265.4.82 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

---

**0.15.265.4.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping (`  
 `const typename Poly::PolyType & _branchingPolynomial,`  
 `const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.265.4.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.265.4.85 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset (`

`ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.265.4.86 `receivedVariable()`** `bool smtrat::Module::receivedVariable (`  
 `carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.265.4.87 `remove()`** `void smtrat::Module::remove (`  
 `ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.265.4.88 `removeCore()`** `template<class Settings >`  
`void smtrat::IntBlastModule< Settings >::removeCore (`  
 `ModuleInput::const_iterator _subformula ) [virtual]`

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.265.4.89 `removeOrigin()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.265.4.90 `removeOrigins()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (`

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
 [inherited]
```

**0.15.265.4.91 `rPassedFormula()`** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const`

`[inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.265.4.92 `rReceivedFormula()`** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const`

`[inline], [inherited]`

#### Returns

A reference to the formula passed to this module.

**0.15.265.4.93 `runBackends() [1/2]`** `virtual Answer smtrat::Module::runBackends ( ) [inline],`

`[protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.265.4.94 `runBackends() [2/2]`** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.265.4.95 setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]  
Sets this modules unique ID to identify itself.

Parameters

|                             |                                    |
|-----------------------------|------------------------------------|
| $\leftrightarrow$<br>$\_id$ | The id to set this module's id to. |
|-----------------------------|------------------------------------|

**0.15.265.4.96 setThreadPriority()** void smtrat::Module::setThreadPriority ( thread\_priority \_threadPriority ) [inline], [inherited]  
Sets the priority of this module to get a thread for running its check procedure.

Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| $\_threadPriority$ | The priority to set this module's thread priority to. |
|--------------------|-------------------------------------------------------|

**0.15.265.4.97 solverState()** Answer smtrat::Module::solverState ( ) const [inline], [inherited]

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.265.4.98 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]  
Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

Parameters

|                       |                                                  |
|-----------------------|--------------------------------------------------|
| $\_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|-----------------------|--------------------------------------------------|

**0.15.265.4.99 threadPriority()** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]

Returns

The priority of this module to get a thread for running its check procedure.

**0.15.265.4.100 updateAllModels()** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]  
Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.265.4.101 updateLemmas()** void smtrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.265.4.102 updateModel()** template<class Settings >  
void smtrat::IntBlastModule< Settings >::updateModel () const [virtual]  
Updates the current assignment into the model.  
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.265.4.103 usedBackends()** const std::vector<[Module](#)\*>& smtrat::Module::usedBackends ()  
const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.265.5 Field Documentation

**0.15.265.5.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected],  
[inherited]

The backends of this module which have been used.

**0.15.265.5.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected],  
[inherited]

Stores all satisfying assignments.

**0.15.265.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer  
[protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.265.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform  
[protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.265.5.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.265.5.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.265.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smtrat::Module::mFirstSubformulaToPass  
[protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.265.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.265.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.265.5.10 mFullCheck** `bool smrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.265.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smrat::Module::mInfeasibleSubsets [protected], [inherited]`  
Stores the infeasible subsets.

**0.15.265.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints [protected], [inherited]`  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.265.5.13 mLastBranches** `std::vector<Branching> smrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`  
Stores the last branches in a cycle buffer.

**0.15.265.5.14 mLemmas** `std::vector<Lemma> smrat::Module::mLemmas [protected], [inherited]`  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.265.5.15 mModel** `Model smrat::Module::mModel [mutable], [protected], [inherited]`  
Stores the assignment of the current satisfiable result, if existent.

**0.15.265.5.16 mModelComputed** `bool smrat::Module::mModelComputed [mutable], [protected], [inherited]`  
True, if the model has already been computed.

**0.15.265.5.17 mNumOfBranchVarsToStore** `std::size_t smrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.265.5.18 mObjectiveVariable** `carl::Variable smrat::Module::mObjectiveVariable [protected], [inherited]`  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.265.5.19 mOldSplittingVariables** `std::vector<FormulaT> smrat::Module::mOldSplittingVariables [static], [inherited]`  
Reusable splitting variables.

**0.15.265.5.20 mpManager** `Manager* const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.265.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter`  
[mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.265.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected],  
[inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.265.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheory←→Propagations` [protected], [inherited]

**0.15.265.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends` [protected],  
[inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.265.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters`  
[protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.266 smtrat::IntEqModule< Settings > Class Template Reference

A module which checks whether the equations contained in the received (linear integer) formula have a solution.  
`#include <IntEqModule.h>`

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
- `IntEqModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~IntEqModule ()`
- `bool addCore (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- `void removeCore (ModuleInput::const_iterator _subformula)`

*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- `void updateModel () const`

*Updates the current assignment into the model.*
- `Answer checkCore ()`

*Checks the received formula for consistency.*
- `bool inform (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- `void deinform (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*

- virtual void `init ()`

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- bool `add (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- virtual `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_← VARIABLE)`

*Checks the received formula for consistency.*
- virtual void `remove (ModuleInput::const_iterator _subformula)`

*Removes everything related to the given sub-formula of the received formula.*
- virtual void `updateAllModels ()`

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > `getModelEqualities () const`

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned `currentlySatisfiedByBackend (const FormulaT &_formula) const`
- virtual unsigned `currentlySatisfied (const FormulaT &) const`
- bool `receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- std::size\_t `id () const`
- void `setId (std::size_t _id)`

*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- void `setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput * pReceivedFormula () const`
- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const std::vector< `Model` > & `allModels () const`
- const std::vector< `FormulaSetT` > & `infeasibleSubsets () const`
- const std::vector< `Module` \* > & `usedBackends () const`
- const carl::FastSet< `FormulaT` > & `constraintsToInform () const`
- const carl::FastSet< `FormulaT` > & `informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

- Stores all lemmas of any backend of this module in its own lemma vector.
- void `collectTheoryPropagations ()`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Prints everything relevant of the solver.*
- virtual std::pair< bool, FormulaT > `getReceivedFormulaSimplified ()`

*Prints the vector of the received formula.*
- void `print (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore = 5`

*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`

*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches = 0`

*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`

*Reusable splitting variables.*

### Protected Member Functions

- virtual bool `informCore (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- virtual void `deinformCore (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- bool `anAnswerFound () const`

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel () const`

*Clears the assignment, if any was found.*
- void `clearModels () const`

- Clears all assignments, if any was found.
- void `cleanModel () const`

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin ()`
- `ModuleInput::iterator passedFormulaEnd ()`
- void `addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`

*Gets the origins of the passed formula at the given position.*
- void `getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
- void `getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- std::pair< `ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
- std::pair< `ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
- void `informBackends (const FormulaT &_constraint)`

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform (const FormulaT &_constraint)`

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula (const FormulaT &_origin) const`
- std::pair< `ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)`

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset ()`

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const`
- void `getInfeasibleSubsets ()`

*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT > getInfeasibleSubsets (const Module &_backend) const`

*Get the infeasible subsets the given backend provides.*
- const `Model & backendsModel () const`

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel () const`
- void `getBackendsAllModels () const`

*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`

- std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const  
*Merges the two vectors of sets into the first one.*
- size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
- bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const  
*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool [branchAt](#) (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)  
*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &&\_premise=std::vector< [FormulaT](#) >())
- bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
- void [splitUnequalConstraint](#) (const [FormulaT](#) &)  
*Adds the following lemmas for the given constraint p!=0.*
- unsigned [checkModel](#) () const
- void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()  
*Gets all InformationRelevantFormulas.*
- bool [isLemmaLevel](#) ([LemmaLevel](#) level)  
*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool [modelsDisjoint](#) (const Model &\_modelA, const Model &\_modelB)  
*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< [FormulaSetT](#) > [mInfeasibleSubsets](#)  
*Stores the infeasible subsets.*
- Manager \*const [mpManager](#)  
*A reference to the manager.*
- Model [mModel](#)  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< Model > [mAllModels](#)  
*Stores all satisfying assignments.*
- bool [mModelComputed](#)  
*True, if the model has already been computed.*
- bool [mFinalCheck](#)  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool [mFullCheck](#)  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable [mObjectiveVariable](#)  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< TheoryPropagation > [mTheoryPropagations](#)

- std::atomic< [Answer](#) > mSolverState  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* mBackendsFoundAnswer  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- [Conditionals](#) mFoundAnswer  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< [Module](#) \* > mUsedBackends  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< [Module](#) \* > mAllBackends  
*The backends of this module which have been used.*
- std::vector< [Lemma](#) > mLemmas  
*Stores the lemmas being valid formulas this module or its backends made.*
- [ModuleInput](#)::iterator mFirstSubformulaToPass  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< [FormulaT](#) > mConstraintsToInform  
*Stores the constraints which the backends must be informed about.*
- carl::FastSet< [FormulaT](#) > mInformedConstraints  
*Stores the position of the first constraint of which no backend has been informed about.*
- [ModuleInput](#)::const\_iterator mFirstUncheckedReceivedSubformula  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned mSmallerMusesCheckCounter  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > mVariableCounters  
*Maps variables to the number of their occurrences.*

### 0.15.266.1 Detailed Description

```
template<typename Settings>
class smrat::IntEqModule< Settings >
```

A module which checks whether the equations contained in the received (linear integer) formula have a solution.

### 0.15.266.2 Member Typedef Documentation

```
0.15.266.2.1 SettingsType template<typename Settings >
typedef Settings smrat::IntEqModule< Settings >::SettingsType
```

### 0.15.266.3 Member Enumeration Documentation

```
0.15.266.3.1 LemmaType enum smrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

#### 0.15.266.4 Constructor & Destructor Documentation

**0.15.266.4.1 `IntEqModule()`** template<typename Settings>  
`smtrat::IntEqModule< Settings >::IntEqModule (`  
 `const ModuleInput * _formula,`  
 `Conditionals & _conditionals,`  
 `Manager * _manager = NULL )`

**0.15.266.4.2 `~IntEqModule()`** template<typename Settings>  
`smtrat::IntEqModule< Settings >::~IntEqModule ( ) [inline]`

#### 0.15.266.5 Member Function Documentation

**0.15.266.5.1 `add()`** bool smtrat::Module::add (  
 `ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.266.5.2 `addConstraintToInform()`** void smtrat::Module::addConstraintToInform (  
 `const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

##### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.266.5.3 `addCore()`** template<typename Settings>  
`bool smtrat::IntEqModule< Settings >::addCore (`  
 `ModuleInput::const_iterator _subformula ) [virtual]`

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.266.5.4 addInformationRelevantFormula()** void smtrat::Module::addInformationRelevantFormula (  
    const [FormulaT](#) & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.266.5.5 addLemma()** void smtrat::Module::addLemma (  
    const [FormulaT](#) & \_lemma,  
    const [LemmaType](#) & \_lt = [LemmaType::NORMAL](#),  
    const [FormulaT](#) & \_preferredFormula = [FormulaT](#)( [carl::FormulaType::TRUE](#) ) ) [inline],  
[inherited]

Stores a lemma being a valid formula.

#### Parameters

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.266.5.6 addOrigin()** void smtrat::Module::addOrigin (  
    [ModuleInput::iterator](#) \_formula,  
    const [FormulaT](#) & \_origin ) [inline], [protected], [inherited]

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

**0.15.266.5.7 addReceivedSubformulaToPassedFormula()** std::pair<[ModuleInput::iterator](#),bool>  
smtrat::Module::addReceivedSubformulaToPassedFormula (  
    [ModuleInput::const\\_iterator](#) \_subformula ) [inline], [protected], [inherited]

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>_subformula</i> | The sub-formula of the received formula to copy. |
|--------------------|--------------------------------------------------|

**0.15.266.5.8 addSubformulaToPassedFormula()** [1/3] std::pair<[ModuleInput::iterator](#),bool> smtrat↔

```
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.266.5.9 `addSubformulaToPassedFormula()` [2/3] std::pair<[ModuleInput::iterator](#),bool> smrat->**

```
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.266.5.10 `addSubformulaToPassedFormula()` [3/3] std::pair<[ModuleInput::iterator](#),bool>**

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.266.5.11 allModels()** const std::vector<[Model](#)>& smtrat::Module::allModels () const [inline], [inherited]

**Returns**

All satisfying assignments, if existent.

**0.15.266.5.12 anAnswerFound()** bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.266.5.13 answerFound()** const [smtrat::Conditionals](#)& smtrat::Module::answerFound () const [inline], [inherited]

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.266.5.14 backendsModel()** const [Model](#) & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.266.5.15 branchAt() [1/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const [Rational](#) & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< [FormulaT](#) > & \_premise = std::vector<[FormulaT](#)>() ) [inline], [protected], [inherited]

**0.15.266.5.16 branchAt() [2/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const [Rational](#) & \_value, std::vector< [FormulaT](#) > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.266.5.17 branchAt() [3/4]** bool smtrat::Module::branchAt ( const [Poly](#) & \_polynomial, bool \_integral, const [Rational](#) & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< [FormulaT](#) > & \_premise = std::vector<[FormulaT](#)>() ) [inline], [protected], [inherited]

**0.15.266.5.18 branchAt() [4/4]** `bool smtrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

**Parameters**

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

**0.15.266.5.19 check() [Answer](#)** `smtrat::Module::check (`

```
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.266.5.20 checkCore()** `template<typename Settings >`

```
Answer smtrat::IntEqModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

**Returns**

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.266.5.21 `checkInfSubsetForMinimality()`** `void smrat::Module::checkInfSubsetForMinimality ( std::vector< FormulaSetT >::const_iterator _infssubset,`  
`const std::string & _filename = "smaller_muses",`  
`unsigned _maxSizeDifference = 1 ) const [inherited]`

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.266.5.22 `checkModel()`** `unsigned smrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.266.5.23 `cleanModel()`** `void smrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.266.5.24 `clearLemmas()`** `void smrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.266.5.25 `clearModel()`** `void smrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.266.5.26 `clearModels()`** `void smrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.266.5.27 `clearPassedFormula()`** `void smrat::Module::clearPassedFormula ( ) [protected], [inherited]`

```
0.15.266.5.28 collectOrigins() [1/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

```
0.15.266.5.29 collectOrigins() [2/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

```
0.15.266.5.30 collectTheoryPropagations() void smtrat::Module::collectTheoryPropagations ()
[inherited]
```

```
0.15.266.5.31 constraintsToInform() const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]
```

#### Returns

The constraints which the backends must be informed about.

```
0.15.266.5.32 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAbstractModule< Settings >](#), [smtrat::LRAbstractModule< LRASettingsICP >](#), and [smtrat::LRAbstractModule< LRASettings >](#)

```
0.15.266.5.33 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.266.5.34 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.266.5.35 `deinformCore()`** `virtual void smtrat::Module::deinformCore (``const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.266.5.36 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (``const std::vector< FormulaT > & origins ) const [protected], [inherited]`**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.266.5.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (``ModuleInput::iterator _subformula,``bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.266.5.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

```
0.15.266.5.39 findBestOrigin() std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

```
0.15.266.5.40 firstSubformulaToPass() ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]
```

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

```
0.15.266.5.41 firstUncheckedReceivedSubformula() ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]
```

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

```
0.15.266.5.42 freeSplittingVariable() static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable) [inline], [static], [inherited]
```

```
0.15.266.5.43 generateTrivialInfeasibleSubset() void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

```
0.15.266.5.44 getBackendsAllModels() void smtrat::Module::getBackendsAllModels () const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

```
0.15.266.5.45 getBackendsModel() void smtrat::Module::getBackendsModel () const [protected], [inherited]
```

```
0.15.266.5.46 getInfeasibleSubsets() [1/2] void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

**0.15.266.5.47 getInfeasibleSubsets()** [2/2] std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets (

    const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.266.5.48 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.266.5.49 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.266.5.50 `getOrigins()` [1/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.266.5.51 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.266.5.52 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.266.5.53 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT >` `smtrat::Module::getReceivedFormulaSimplified( )` [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.266.5.54 `hasLemmas()`** `bool smtrat::Module::hasLemmas( )` [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.266.5.55 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const` [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.266.5.56 `id()`** `std::size_t smtrat::Module::id( ) const` [inline], [inherited]

**Returns**

A unique ID to identify this module instance.

**0.15.266.5.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const` [inline], [inherited]

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.266.5.58 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint )` [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before `assertSubformula` is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.266.5.59 informBackends()** void smrat::Module::informBackends ( const FormulaT & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**0.15.266.5.60 informCore()** virtual bool smrat::Module::informCore (

const FormulaT & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.266.5.61 informedConstraints()** const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.266.5.62 init()** void smrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

**0.15.266.5.63 is\_minimizing()** bool smrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.266.5.64 isLemmaLevel()** `bool smtrat::Module::isLemmaLevel (``LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.266.5.65 isPreprocessor()** `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.266.5.66 lemmas()** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.266.5.67 merge()** `std::vector< FormulaT > smtrat::Module::merge (``const std::vector< FormulaT > & _vecSetA,``const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.266.5.68 model()** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.266.5.69 modelsDisjoint()** `bool smtrat::Module::modelsDisjoint (``const Model & _modelA,``const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.266.5.70 `moduleName()`** `template<typename Settings >`  
`std::string smtrat::IntEqModule< Settings >::moduleName ( ) const [inline], [virtual]`

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.266.5.71 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.266.5.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`  
`const FormulaT & _origin ) const [protected], [inherited]`

**0.15.266.5.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin (`  
`) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.266.5.74 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
`[inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.266.5.75 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.266.5.76 `pReceivedFormula()`** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.266.5.77 `print()`** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.266.5.78 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.266.5.79 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.266.5.80 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.266.5.81 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.266.5.82 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.266.5.83 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.266.5.84 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline], [inherited]
```

Notifies that the received formulas has been checked.

```
0.15.266.5.85 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs< InfeasibleSubset (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.266.5.86 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.266.5.87 remove() void smtrat::Module::remove (
```

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

```
0.15.266.5.88 removeCore() template<typename Settings >
void smtrat::IntEqModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.266.5.89 `removeOrigin()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (`

`ModuleInput::iterator _formula,`  
`const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.266.5.90 `removeOrigins()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (`

`ModuleInput::iterator _formula,`  
`const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],`  
`[inherited]`

**0.15.266.5.91 `rPassedFormula()`** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const`  
[inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.266.5.92 `rReceivedFormula()`** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const`  
[inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.266.5.93 `runBackends() [1/2]`** `virtual Answer smtrat::Module::runBackends ( ) [inline],`  
[protected], [virtual], [inherited]

Reimplemented in [smtrat::PModule](#).

**0.15.266.5.94 `runBackends() [2/2]`** `Answer smtrat::Module::runBackends (`

`bool _final,`  
`bool _full,`  
`carl::Variable _objective ) [protected], [virtual], [inherited]`

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.266.5.95 setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]  
 Sets this modules unique ID to identify itself.

#### Parameters

|                                                 |                                    |
|-------------------------------------------------|------------------------------------|
| $\leftarrow$<br>$\overline{\leftarrow}$<br>$id$ | The id to set this module's id to. |
|-------------------------------------------------|------------------------------------|

**0.15.266.5.96 setThreadPriority()** void smtrat::Module::setThreadPriority ( thread\_priority \_threadPriority ) [inline], [inherited]  
 Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| $_threadPriority$ | The priority to set this module's thread priority to. |
|-------------------|-------------------------------------------------------|

**0.15.266.5.97 solverState()** Answer smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.266.5.98 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]  
 Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                      |                                                  |
|----------------------|--------------------------------------------------|
| $_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|----------------------|--------------------------------------------------|

**0.15.266.5.99 threadPriority()** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.266.5.100 updateAllModels()** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]  
 Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
 Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.266.5.101 updateLemmas()** void smtrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.266.5.102 updateModel()** template<typename Settings >  
void smtrat::IntEqModule< Settings >::updateModel () const [virtual]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.266.5.103 usedBackends()** const std::vector<[Module](#)\*>& smtrat::Module::usedBackends ()  
const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.266.6 Field Documentation

**0.15.266.6.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected],  
[inherited]

The backends of this module which have been used.

**0.15.266.6.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected],  
[inherited]

Stores all satisfying assignments.

**0.15.266.6.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer  
[protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.266.6.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform  
[protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.266.6.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.266.6.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.266.6.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smtrat::Module::mFirstSubformulaToPass  
[protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.266.6.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.266.6.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.266.6.10 mFullCheck** `bool smtrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.266.6.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`  
Stores the infeasible subsets.

**0.15.266.6.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.266.6.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`  
Stores the last branches in a cycle buffer.

**0.15.266.6.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.266.6.15 mModel** `Model smtrat::Module::mModel [mutable], [protected], [inherited]`  
Stores the assignment of the current satisfiable result, if existent.

**0.15.266.6.16 mModelComputed** `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`  
True, if the model has already been computed.

**0.15.266.6.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.266.6.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.266.6.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables [static], [inherited]`  
Reusable splitting variables.

**0.15.266.6.20 mpManager** `Manager* const smtrat::Module::mpManager [protected], [inherited]`  
A reference to the manager.

**0.15.266.6.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.266.6.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.266.6.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

**0.15.266.6.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.266.6.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`  
Maps variables to the number of their occurrences.

## 0.15.267 smtrat::mcsat::icp::IntervalPropagation Class Reference

```
#include <IntervalPropagation.h>
```

### Public Member Functions

- `IntervalPropagation (const std::vector< carl::Variable > &vars, const std::vector< FormulaT > &constraints, const Model &model)`
- `std::optional< FormulaT > execute ()`

### 0.15.267.1 Constructor & Destructor Documentation

**0.15.267.1.1 IntervalPropagation()** `smtrat::mcsat::icp::IntervalPropagation::IntervalPropagation (const std::vector< carl::Variable > & vars, const std::vector< FormulaT > & constraints, const Model & model ) [inline]`

### 0.15.267.2 Member Function Documentation

**0.15.267.2.1 execute()** `std::optional<FormulaT> smtrat::mcsat::icp::IntervalPropagation::execute () [inline]`

## 0.15.268 Minisat::IntOption Class Reference

```
#include <Options.h>
```

## Public Member Functions

- `IntOption` (const char \*c, const char \*n, const char \*d, int32\_t def=int32\_t(), `IntRange` r=`IntRange`(INT\_MIN, INT\_MAX))
- `operator int32_t` (void) const
- `IntOption & operator=` (int32\_t x)
- virtual void `help` (bool verbose=false)

## Static Protected Member Functions

- static `vec< Option * >` & `getOptionList` ()
- static const char \*& `getUsageString` ()
- static const char \*& `getHelpPrefixString` ()

## Protected Attributes

- `IntRange range`
- `int32_t value`
- const char \* `name`
- const char \* `description`
- const char \* `category`
- const char \* `type_name`

### 0.15.268.1 Constructor & Destructor Documentation

#### 0.15.268.1.1 `IntOption()` Minisat::IntOption::IntOption (

```
 const char * c,
 const char * n,
 const char * d,
 int32_t def = int32_t(),
 IntRange r = IntRange(INT_MIN, INT_MAX)) [inline]
```

### 0.15.268.2 Member Function Documentation

#### 0.15.268.2.1 `getHelpPrefixString()` static const char\*& Minisat::Option::getHelpPrefixString ( ) [inline], [static], [protected], [inherited]

#### 0.15.268.2.2 `getOptionList()` static `vec<Option*>&` Minisat::Option::getOptionList ( ) [inline], [static], [protected], [inherited]

#### 0.15.268.2.3 `getUsageString()` static const char\*& Minisat::Option::getUsageString ( ) [inline], [static], [protected], [inherited]

#### 0.15.268.2.4 `help()` virtual void Minisat::IntOption::help ( bool verbose = false ) [inline], [virtual] Implements `Minisat::Option`.

#### 0.15.268.2.5 `operator int32_t()` Minisat::IntOption::operator int32\_t ( void ) const [inline]

**0.15.268.2.6 operator=()** `IntOption& Minisat::IntOption::operator= ( int32_t x ) [inline]`

### 0.15.268.3 Field Documentation

**0.15.268.3.1 category** `const char* Minisat::Option::category [protected], [inherited]`

**0.15.268.3.2 description** `const char* Minisat::Option::description [protected], [inherited]`

**0.15.268.3.3 name** `const char* Minisat::Option::name [protected], [inherited]`

**0.15.268.3.4 range** `IntRange Minisat::IntOption::range [protected]`

**0.15.268.3.5 type\_name** `const char* Minisat::Option::type_name [protected], [inherited]`

**0.15.268.3.6 value** `int32_t Minisat::IntOption::value [protected]`

## 0.15.269 Minisat::IntRange Struct Reference

#include <Options.h>

### Public Member Functions

- **IntRange** (int b, int e)

### Data Fields

- int **begin**
- int **end**

### 0.15.269.1 Constructor & Destructor Documentation

**0.15.269.1.1 IntRange()** `Minisat::IntRange::IntRange ( int b, int e ) [inline]`

### 0.15.269.2 Field Documentation

**0.15.269.2.1 begin** `int Minisat::IntRange::begin`

**0.15.269.2.2 end** `int Minisat::IntRange::end`

## 0.15.270 smrat::is\_sample\_outside< S > Struct Template Reference

#include <Sampling.h>

## Static Public Member Functions

- template<typename T >  
static bool **is\_outside** (const **cadcells::RAN** &sample, const std::set< **cadcells::datastructures::SampledDerivationRef<** T **>, SampledDerivationRefCompare >** &derivations)

### 0.15.270.1 Member Function Documentation

**0.15.270.1.1 **is\_outside()**** template<IsSampleOutsideAlgorithm S>  
template<typename T >  
static bool **smtrat::is\_sample\_outside< S >::is\_outside** (  
    const **cadcells::RAN** & sample,  
    const std::set< **cadcells::datastructures::SampledDerivationRef<** T **>, SampledDerivationRefCompare >** & derivations ) [static]

## 0.15.271 **smtrat::is\_sample\_outside< IsSampleOutsideAlgorithm::DEFAULT > Struct Reference**

#include <Sampling.h>

## Static Public Member Functions

- template<typename T >  
static bool **is\_outside** (const **cadcells::RAN** &sample, const std::set< **cadcells::datastructures::SampledDerivationRef<** T **>, SampledDerivationRefCompare >** &derivations)

### 0.15.271.1 Member Function Documentation

**0.15.271.1.1 **is\_outside()**** template<typename T >  
static bool **smtrat::is\_sample\_outside< IsSampleOutsideAlgorithm::DEFAULT >::is\_outside** (  
    const **cadcells::RAN** & sample,  
    const std::set< **cadcells::datastructures::SampledDerivationRef<** T **>, SampledDerivationRefCompare >** & derivations ) [inline], [static]

## 0.15.272 **smtrat::is\_variant< T > Struct Template Reference**

States whether a given type is a boost::variant.

#include <VariantMap.h>

### 0.15.272.1 Detailed Description

```
template<class T>
struct smtrat::is_variant< T >
```

States whether a given type is a boost::variant.

By default, a type is not.

## 0.15.273 **smtrat::expression::ITEExpression Struct Reference**

#include <ExpressionContent.h>

## Public Member Functions

- **ITEExpression** (ITEType, Expression &&\_condition, Expression &&\_then, Expression &&\_else)

**Data Fields**

- `ITEType type`
- `Expression condition`
- `Expression thencase`
- `Expression elsecase`

**0.15.273.1 Constructor & Destructor Documentation**

```
0.15.273.1.1 ITEExpression() smtrat::expression::ITEExpression::ITEExpression (
 ITEType ,
 Expression && _condition,
 Expression && _then,
 Expression && _else) [inline]
```

**0.15.273.2 Field Documentation**

**0.15.273.2.1 condition** `Expression` smtrat::expression::ITEExpression::condition

**0.15.273.2.2 elsecase** `Expression` smtrat::expression::ITEExpression::elsecase

**0.15.273.2.3 thencase** `Expression` smtrat::expression::ITEExpression::thencase

**0.15.273.2.4 type** `ITEType` smtrat::expression::ITEExpression::type

**0.15.274 benchmax::RandomizationAdaptor< T >::iterator Struct Reference**

```
#include <Jobs.h>
```

**Public Member Functions**

- `iterator (std::size_t k, std::size_t factor, const std::vector< T > &data)`  
*Create a randomized iterator.*
- `const T & operator* ()`  
*Dereference iterator.*
- `iterator & operator++ ()`  
*Increment iterator.*
- `bool operator!= (const iterator &rhs) const`  
*Compare two iterators.*

**0.15.274.1 Constructor & Destructor Documentation**

```
0.15.274.1.1 iterator() template<typename T >
benchmax::RandomizationAdaptor< T >::iterator::iterator (
 std::size_t k,
 std::size_t factor,
 const std::vector< T > & data) [inline]
```

Create a randomized iterator.

### 0.15.274.2 Member Function Documentation

**0.15.274.2.1 operator"!=()** template<typename T >  
 bool **benchmax::RandomizationAdaptor**< T >::iterator::operator!= ( const iterator & rhs ) const [inline]

Compare two iterators.

**0.15.274.2.2 operator\*()** template<typename T >  
 const T& **benchmax::RandomizationAdaptor**< T >::iterator::operator\* ( ) [inline]

Dereference iterator.

**0.15.274.2.3 operator++()** template<typename T >  
 iterator& **benchmax::RandomizationAdaptor**< T >::iterator::operator++ ( ) [inline]

Increment iterator.

## 0.15.275 smtrat::ModuleInput::IteratorCompare Struct Reference

#include <ModuleInput.h>

### Public Member Functions

- bool **operator()** (const iterator i1, const iterator i2) const

### 0.15.275.1 Member Function Documentation

**0.15.275.1.1 operator()** bool smtrat::ModuleInput::IteratorCompare::operator() ( const\_iterator i1, const\_iterator i2 ) const [inline]

## 0.15.276 benchmax::Jobs Class Reference

Represents a set of jobs, constructed as the cartesian product of a set of tools and a set of benchmarks.

#include <Jobs.h>

### Public Member Functions

- **Jobs** (const Tools &tools, const BenchmarkSet &benchmarks)  
*Construct jobs from a set of tools and a set of benchmark files.*
- const auto & **tools** () const  
*Returns the set of tools.*
- const auto & **files** () const  
*Returns the set of files.*
- auto **size** () const  
*Returns the overall number of jobs.*
- auto **begin** () const  
*Begin iterator.*
- auto **end** () const  
*End iterator.*
- auto **randomized** () const  
*Returns all jobs in a pseudo-randomized order.*

### 0.15.276.1 Detailed Description

Represents a set of jobs, constructed as the cartesian product of a set of tools and a set of benchmarks.

### 0.15.276.2 Constructor & Destructor Documentation

```
0.15.276.2.1 Jobs() benchmax::Jobs::Jobs (
 const Tools & tools,
 const BenchmarkSet & benchmarks) [inline]
```

Construct jobs from a set of tools and a set of benchmark files.

### 0.15.276.3 Member Function Documentation

```
0.15.276.3.1 begin() auto benchmax::Jobs::begin () const [inline]
```

Begin iterator.

```
0.15.276.3.2 end() auto benchmax::Jobs::end () const [inline]
```

End iterator.

```
0.15.276.3.3 files() const auto& benchmax::Jobs::files () const [inline]
```

Returns the set of files.

```
0.15.276.3.4 randomized() auto benchmax::Jobs::randomized () const [inline]
```

Returns all jobs in a pseudo-randomized order.

```
0.15.276.3.5 size() auto benchmax::Jobs::size () const [inline]
```

Returns the overall number of jobs.

```
0.15.276.3.6 tools() const auto& benchmax::Jobs::tools () const [inline]
```

Returns the set of tools.

## 0.15.277 smtrat::JunctorMerger Struct Reference

```
#include <JunctorMerger.h>
```

### Public Member Functions

- `FormulaT rewrite_or (const FormulaT &formula, FormulaSetT &&subformulas)`
- `FormulaT rewrite_and (const FormulaT &formula, FormulaSetT &&subformulas)`
- `FormulaT rewrite_xor (const FormulaT &formula, FormulasMultiT &&subformulas)`

### 0.15.277.1 Member Function Documentation

```
0.15.277.1.1 rewrite_and() FormulaT smtrat::JunctorMerger::rewrite_and (
```

```
 const FormulaT & formula,
 FormulaSetT && subformulas) [inline]
```

```
0.15.277.1.2 rewrite_or() FormulaT smtrat::JunctorMerger::rewrite_or (
 const FormulaT & formula,
 FormulaSetT && subformulas) [inline]
```

```
0.15.277.1.3 rewrite_xor() FormulaT smtrat::JunctorMerger::rewrite_xor (
 const FormulaT & formula,
 FormulasMultiT && subformulas) [inline]
```

## 0.15.278 smtrat::parser::KeywordParser Struct Reference

Parses keywords: :simple\_symbol  
`#include <Lexicon.h>`

### Public Member Functions

- `KeywordParser ()`

### Data Fields

- `qi::rule< Iterator, std::string(), Skipper > main`
- `SimpleSymbolParser simple`

### 0.15.278.1 Detailed Description

Parses keywords: :simple\_symbol

### 0.15.278.2 Constructor & Destructor Documentation

```
0.15.278.2.1 KeywordParser() smtrat::parser::KeywordParser::KeywordParser () [inline]
```

### 0.15.278.3 Field Documentation

```
0.15.278.3.1 main qi::rule<Iterator, std::string(), Skipper> smtrat::parser::KeywordParser::main
```

```
0.15.278.3.2 simple SimpleSymbolParser smtrat::parser::KeywordParser::simple
```

## 0.15.279 Minisat::lbool Class Reference

```
#include <SolverTypes.h>
```

### Public Member Functions

- `lbool (uint8_t v)`
- `lbool ()`
- `lbool (bool x)`
- `bool operator== (lbool b) const`
- `bool operator!= (lbool b) const`
- `lbool operator^ (bool b) const`
- `lbool operator&& (lbool b) const`
- `lbool operator|| (lbool b) const`

**Friends**

- int `toInt (lbool l)`
- `lbool toLbool (int v)`

**0.15.279.1 Constructor & Destructor Documentation**

**0.15.279.1.1 `lbool()` [1/3]** Minisat:::lbool:::lbool ( uint8\_t v ) [inline], [explicit]

**0.15.279.1.2 `lbool()` [2/3]** Minisat:::lbool:::lbool ( ) [inline]

**0.15.279.1.3 `lbool()` [3/3]** Minisat:::lbool:::lbool ( bool x ) [inline], [explicit]

**0.15.279.2 Member Function Documentation**

**0.15.279.2.1 `operator"!="()`** bool Minisat:::lbool:::operator!= ( lbool b ) const [inline]

**0.15.279.2.2 `operator&&()`** lbool Minisat:::lbool:::operator&& ( lbool b ) const [inline]

**0.15.279.2.3 `operator==()`** bool Minisat:::lbool:::operator== ( lbool b ) const [inline]

**0.15.279.2.4 `operator^()`** lbool Minisat:::lbool:::operator^ ( bool b ) const [inline]

**0.15.279.2.5 `operator"||"()`** lbool Minisat:::lbool:::operator|| ( lbool b ) const [inline]

**0.15.279.3 Friends And Related Function Documentation**

**0.15.279.3.1 `toInt`** int toInt ( lbool l ) [friend]

**0.15.279.3.2 `toLbool`** lbool toLbool ( int v ) [friend]

**0.15.280 smtrat::mcsat::onecell::LDBFilteredAllSelectiveSettings Struct Reference**

```
#include <onecell.h>
```

**Static Public Attributes**

- constexpr static auto `cell_heuristic` = `cadcells::representation::LOWEST_DEGREE_BARRIERS_EW`
- constexpr static auto `covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING_EW`
- constexpr static auto `op` = `cadcells::operators::op::mccallum_filtered_all_selective`

**0.15.280.1 Field Documentation**

**0.15.280.1.1 `cell_heuristic`** constexpr static auto `smtrat::mcsat::onecell::LDBFilteredAllSelective`→  
Settings::`cell_heuristic` = `cadcells::representation::LOWEST_DEGREE_BARRIERS_EW` [static],  
[constexpr]

**0.15.280.1.2 `covering_heuristic`** constexpr static auto `smtrat::mcsat::onecell::LDBFilteredAllSelective`→  
Settings::`covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING_EW` [static], [constexpr]

**0.15.280.1.3 `op`** constexpr static auto `smtrat::mcsat::onecell::LDBFilteredAllSelective`→  
Settings::`op` = `cadcells::operators::op::mccallum_filtered_all_selective` [static], [constexpr]

**0.15.281 smtrat::mcsat::onecell::LDBSettings Struct Reference**

```
#include <onecell.h>
```

**Static Public Attributes**

- constexpr static auto `cell_heuristic` = `cadcells::representation::LOWEST_DEGREE_BARRIERS`
- constexpr static auto `covering_heuristic` = `cadcells::representation::BIGGEST_CELL_COVERING`
- constexpr static auto `op` = `cadcells::operators::op::mccallum`

**0.15.281.1 Field Documentation**

**0.15.281.1.1 `cell_heuristic`** constexpr static auto `smtrat::mcsat::onecell::LDBSettings::cell`→  
heuristic = `cadcells::representation::LOWEST_DEGREE_BARRIERS` [static], [constexpr]

**0.15.281.1.2 `covering_heuristic`** constexpr static auto `smtrat::mcsat::onecell::LDBSettings::covering`→  
heuristic = `cadcells::representation::BIGGEST_CELL_COVERING` [static], [constexpr]

**0.15.281.1.3 `op`** constexpr static auto `smtrat::mcsat::onecell::LDBSettings::op` = `cadcells::operators::op::mccallum` [static], [constexpr]

**0.15.282 smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound Struct Reference**

```
#include <Tableau.h>
```

## Public Member Functions

- `LearnedBound ()=delete`
- `LearnedBound (const LearnedBound &)=delete`
- `LearnedBound (LearnedBound && _toMove)`
- `LearnedBound (const Value< T1 > & _limit, const Bound< T1, T2 > * _newBound, typename Bound< T1, T2 >::BoundSet::const_iterator _nextWeakerBound, std::vector< const Bound< T1, T2 > * > && _premise)`
- `~LearnedBound ()`

## Data Fields

- `const Bound< T1, T2 > * newBound`
- `Value< T1 > newLimit`
- `Bound< T1, T2 >::BoundSet::const_iterator nextWeakerBound`
- `std::vector< const Bound< T1, T2 > * > premise`

### 0.15.282.1 Constructor & Destructor Documentation

**0.15.282.1.1 LearnedBound() [1/4]** template<class Settings , typename T1 , typename T2 >  
`smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound ( ) [delete]`

**0.15.282.1.2 LearnedBound() [2/4]** template<class Settings , typename T1 , typename T2 >  
`smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound (`  
`const LearnedBound & ) [delete]`

**0.15.282.1.3 LearnedBound() [3/4]** template<class Settings , typename T1 , typename T2 >  
`smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound (`  
`LearnedBound && _toMove ) [inline]`

**0.15.282.1.4 LearnedBound() [4/4]** template<class Settings , typename T1 , typename T2 >  
`smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::LearnedBound (`  
`const Value< T1 > & _limit,`  
`const Bound< T1, T2 > * _newBound,`  
`typename Bound< T1, T2 >::BoundSet::const_iterator _nextWeakerBound,`  
`std::vector< const Bound< T1, T2 > * > && _premise ) [inline]`

**0.15.282.1.5 ~LearnedBound()** template<class Settings , typename T1 , typename T2 >  
`smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::~LearnedBound ( ) [inline]`

### 0.15.282.2 Field Documentation

**0.15.282.2.1 newBound** template<class Settings , typename T1 , typename T2 >  
`const Bound<T1, T2>* smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::newBound`

**0.15.282.2.2 newLimit** template<class Settings , typename T1 , typename T2 >  
`Value<T1> smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound::newLimit`

```
0.15.282.2.3 nextWeakerBound template<class Settings , typename T1 , typename T2 >
Bound<T1, T2>::BoundSet::const_iterator smtrat::lra::Tableau< Settings, T1, T2 >::Learned←
Bound::nextWeakerBound
```

```
0.15.282.2.4 premise template<class Settings , typename T1 , typename T2 >
std::vector< const Bound<T1, T2>*> smtrat::lra::Tableau< Settings, T1, T2 >::LearnedBound←
::premise
```

## 0.15.283 smtrat::Module::Lemma Struct Reference

```
#include <Module.h>
```

### Public Member Functions

- **Lemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lemmaType, const **FormulaT** &\_preferredFormula)  
*Constructor.*

### Data Fields

- **FormulaT mLemma**  
*The lemma to learn.*
- **LemmaType mLemmaType**  
*The type of the lemma.*
- **FormulaT mPreferredFormula**  
*The formula within the lemma, which should be assigned to true in the next decision.*

### 0.15.283.1 Constructor & Destructor Documentation

```
0.15.283.1.1 Lemma() smtrat::Module::Lemma::Lemma (
 const FormulaT & _lemma,
 const LemmaType & _lemmaType,
 const FormulaT & _preferredFormula) [inline]
```

Constructor.

#### Parameters

|                          |                                                                                      |
|--------------------------|--------------------------------------------------------------------------------------|
| <b>_lemma</b>            | The lemma to learn.                                                                  |
| <b>_lemmaType</b>        | The type of the lemma.                                                               |
| <b>_preferredFormula</b> | The formula within the lemma, which should be assigned to true in the next decision. |

### 0.15.283.2 Field Documentation

```
0.15.283.2.1 mLemma FormulaT smtrat::Module::Lemma::mLemma
The lemma to learn.
```

```
0.15.283.2.2 mLemmaType LemmaType smtrat::Module::Lemma::mLemmaType
The type of the lemma.
```

**0.15.283.2.3 mPreferredFormula** `FormulaT smrat::Module::Lemma::mPreferredFormula`  
The formula within the lemma, which should be assigned to true in the next decision.

## 0.15.284 Minisat::LessThan\_default< T > Struct Template Reference

```
#include <Sort.h>
```

### Public Member Functions

- bool `operator()` (`T x, T y`)

#### 0.15.284.1 Member Function Documentation

**0.15.284.1.1 operator()** `template<class T >`  
`bool Minisat::LessThan_default< T >::operator() (`  
    `T x,`  
    `T y ) [inline]`

## 0.15.285 smrat::cad::projection\_compare::level Struct Reference

```
#include <ProjectionComparator.h>
```

## 0.15.286 smrat::cad::sample\_compare::level Struct Reference

```
#include <SampleComparator.h>
```

## 0.15.287 smrat::cad::ProjectionLevelInformation::LevelInfo Struct Reference

```
#include <ProjectionInformation.h>
```

### Public Member Functions

- bool `isBound` (`std::size_t pid`) const
- void `setBound` (`std::size_t pid, bool isBound`)
- bool `isEvaluated` (`std::size_t pid`) const
- void `setEvaluated` (`std::size_t pid, bool isEvaluated`)
- bool `isPurged` (`std::size_t pid`) const
- void `setPurged` (`std::size_t pid, bool isPurged`)
- void `removePurgedFromEvaluated` ()
- void `restrictEvaluatedToPurged` ()

### Data Fields

- `carl::Bitset bounds`  
*Which polynomials are bounds.*
- `carl::Bitset evaluated`  
*Which polynomials have been evaluated w.r.t. purging.*
- `carl::Bitset purged`  
*Which polynomials are purged from the projection. (usually due to bounds)*
- `EquationalConstraints ecs`  
*Equational constraints.*

#### 0.15.287.1 Member Function Documentation

**0.15.287.1.1 `isBound()`** bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isBound ( std::size\_t pid ) const [inline]

**0.15.287.1.2 `isEvaluated()`** bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isEvaluated ( std::size\_t pid ) const [inline]

**0.15.287.1.3 `isPurged()`** bool smtrat::cad::ProjectionLevelInformation::LevelInfo::isPurged ( std::size\_t pid ) const [inline]

**0.15.287.1.4 `removePurgedFromEvaluated()`** void smtrat::cad::ProjectionLevelInformation::LevelInfo::removePurgedFromEvaluated ( ) [inline]

**0.15.287.1.5 `restrictEvaluatedToPurged()`** void smtrat::cad::ProjectionLevelInformation::LevelInfo::restrictEvaluatedToPurged ( ) [inline]

**0.15.287.1.6 `setBound()`** void smtrat::cad::ProjectionLevelInformation::LevelInfo::setBound ( std::size\_t pid, bool isBound ) [inline]

**0.15.287.1.7 `setEvaluated()`** void smtrat::cad::ProjectionLevelInformation::LevelInfo::setEvaluated ( std::size\_t pid, bool isEvaluated ) [inline]

**0.15.287.1.8 `setPurged()`** void smtrat::cad::ProjectionLevelInformation::LevelInfo::setPurged ( std::size\_t pid, bool isPurged ) [inline]

## 0.15.287.2 Field Documentation

**0.15.287.2.1 `bounds`** carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::bounds  
Which polynomials are bounds.

**0.15.287.2.2 `ecs`** EquationalConstraints smtrat::cad::ProjectionLevelInformation::LevelInfo::ecs  
Equational constraints.

**0.15.287.2.3 `evaluated`** carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::evaluated  
Which polynomials have been evaluated w.r.t. purging.

**0.15.287.2.4 `purged`** carl::Bitset smtrat::cad::ProjectionLevelInformation::LevelInfo::purged  
Which polynomials are purged from the projection. (usually due to bounds)

## 0.15.288 smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD Class Reference

```
#include <OneCellCAD.h>
```

### Public Member Functions

- std::optional< CADCell > **constructCADCellEnclosingPoint** (std::vector< std::vector< TagPoly >> &polys, int sectionHeuristic, int sectorHeuristic)
 

*Construct a single CADCell that contains the given 'point' and is sign-invariant for the given polynomials in 'polys'.*
- OneCellCAD (const std::vector< carl::Variable > &variableOrder, const RealAlgebraicPoint< Rational > &point)
 

*Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.*
- std::map< carl::Variable, RAN > **prefixPointToStdMap** (const std::size\_t componentCount)
 

*Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.*
- std::vector< RAN > **isolateLastVariableRoots** (const std::size\_t polyLevel, const Poly &poly)
 

*src/lib/datastructures/mcsat/onecellcad/OneCellCAD.h Given a poly  $p(x_1, \dots, x_n)$ , return all roots of the univariate poly  $p(a_1, \dots, a_{n-1}, x_n)$ , where  $a_1, \dots, a_{n-1}$  are the first algebraic real components from 'point' (SORTED!).*
- bool **vanishesEarly** (const std::size\_t polyLevel, const Poly &poly)
 

*Check if an n-variate 'poly'  $p(x_1, \dots, x_n)$  with  $n \geq 1$  already becomes the zero poly after plugging in  $p(a_1, \dots, a_{n-1}, x_n)$ , where  $a_1, \dots, a_{n-1}$  are the first  $n-1$  algebraic real components from 'point'.*
- bool **isPointRootOfPoly** (const std::size\_t polyLevel, const Poly &poly)
 

*Variables can also be indexed by level.*
- bool **isPointRootOfPoly** (const TagPoly &poly)
 

*Variables can also be indexed by level.*
- std::optional< Poly > **coeffNonNull** (const TagPoly &boundCandidate)
 

*Variables can also be indexed by level.*
- bool **isMainPointInsideCell** (const CADCell &cell)

### Data Fields

- const std::vector< carl::Variable > & **variableOrder**

*Variables can also be indexed by level.*
- const RealAlgebraicPoint< Rational > & **point**

#### 0.15.288.1 Member Function Documentation

**0.15.288.1.1 coeffNonNull()** std::optional<Poly> smtrat::mcsat::onecellcad::OneCellCAD::coeffNonNull (const TagPoly & boundCandidate ) [inline], [inherited]

**0.15.288.1.2 constructCADCellEnclosingPoint()** std::optional<CADCell> smtrat::mcsat::onecellcad::levelwise::LevelwiseCAD::constructCADCellEnclosingPoint ( std::vector< std::vector< TagPoly >> & polys, int sectionHeuristic, int sectorHeuristic ) [inline]

Construct a single CADCell that contains the given 'point' and is sign-invariant for the given polynomials in 'polys'. The construction fails if a polynomial vanishes ( $p(a_1, \dots, a_{i-1}, x_i)$ ). Note that this cell is cylindrical only with respect to the given 'variableOrder'.

#### Parameters

|                      |                                                                                                                            |
|----------------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>variableOrder</i> | must contain unique variables and at least one, because constant polynomials (without a variable) are prohibited.          |
| <i>point</i>         | <i>point.size() &gt;= variables.size()</i> .                                                                               |
| <i>polys</i>         | must contain only non-constant, irreducible tagged polynomials that mention only variables that appear in 'variableOrder'. |

Current level is a [Section](#)  
 Projection part for section case  
 Current level is a [Sector](#)  
 Projection part for sector case  
 optimize memory  
 optimize memory

**0.15.288.1.3 `isMainPointInsideCell()`** `bool smtrat::mcsat::onecellcad::OneCellCAD::isMainPoint←InsideCell (`  
 `const CADCell & cell ) [inline], [inherited]`

**0.15.288.1.4 `isolateLastVariableRoots()`** `std::vector<RAN> smtrat::mcsat::onecellcad::OneCellCAD:::isolateLastVariableRoots (`  
 `const std::size_t polyLevel,`  
 `const Poly & poly ) [inline], [inherited]`  
 src/lib/datastructures/mcsat/onecellcad/OneCellCAD.h Given a poly  $p(x_1, \dots, x_n)$ , return all roots of the univariate poly  $p(a_1, \dots, a_{n-1}, x_n)$ , where  $a_1, \dots, a_{n-1}$  are the first algebraic real components from 'point' (SORTED!).

**0.15.288.1.5 `isPointRootOfPoly() [1/2]`** `bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRoot←OfPoly (`  
 `const std::size_t polyLevel,`  
 `const Poly & poly ) [inline], [inherited]`

**0.15.288.1.6 `isPointRootOfPoly() [2/2]`** `bool smtrat::mcsat::onecellcad::OneCellCAD::isPointRoot←OfPoly (`  
 `const TagPoly & poly ) [inline], [inherited]`

**0.15.288.1.7 `OneCellCAD()`** `smtrat::mcsat::onecellcad::OneCellCAD::OneCellCAD [inline]`

**0.15.288.1.8 `prefixPointToStdMap()`** `std::map<carl::Variable, RAN> smtrat::mcsat::onecellcad:::OneCellCAD::prefixPointToStdMap (`  
 `const std::size_t componentCount ) [inline], [inherited]`  
 Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.

**0.15.288.1.9 `vanishesEarly()`** `bool smtrat::mcsat::onecellcad::OneCellCAD::vanishesEarly (`  
 `const std::size_t polyLevel,`  
 `const Poly & poly ) [inline], [inherited]`

Check if an n-variate 'poly'  $p(x_1, \dots, x_n)$  with  $n \geq 1$  already becomes the zero poly after plugging in  $p(a_1, \dots, a_{n-1}, x_n)$ , where  $a_1, \dots, a_{n-1}$  are the first  $n-1$  algebraic real components from 'point'.

## 0.15.288.2 Field Documentation

**0.15.288.2.1 `point`** `const RealAlgebraicPoint<Rational>& smtrat::mcsat::onecellcad::OneCellCAD:::point [inherited]`

**0.15.288.2.2 variableOrder** const std::vector<carl::Variable>& smrat::mcsat::onecellcad::OneCellCAD::variableOrder [inherited]  
 Variables can also be indexed by level.  
 Polys with mathematical level 1 contain the variable in variableOrder[0]

## 0.15.289 smrat::LevelWiseInformation< Settings > Class Template Reference

```
#include <LevelWiseInformation.h>
```

### Public Member Functions

- `LevelWiseInformation ()`
- `LevelWiseInformation (LevelWiseInformation &&other)`
- `void addDerivation (const datastructures::SampledDerivationRef< PropSet > &derivation)`
- `void addDerivation (datastructures::SampledDerivationRef< PropSet > &&derivation)`
- `const std::set< datastructures::SampledDerivationRef< PropSet >, SampledDerivationRefCompare > & getDerivations () const`
- `void clear ()`
- `bool computeCovering ()`
- `const cadcells::RAN & getSampleOutside () const`
- `bool isPartialCovering () const`
- `bool isFullCovering () const`
- `bool isUnknownCovering () const`
- `bool isFailedCovering () const`
- `CoveringStatus getCoveringStatus () const`
- `LevelWiseInformation & operator= (LevelWiseInformation &&other)`
- `void setDerivations (std::vector< datastructures::SampledDerivationRef< PropSet >> &&derivations)`
- `void removeDerivation (const datastructures::SampledDerivationRef< PropSet > &derivation)`
- `void removeDerivation (const std::vector< datastructures::SampledDerivationRef< PropSet >> &derivations)`
- `void removeConstraint (const cadcells::Constraint &constraint, const std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint >> &derivationConstraints)`
- `std::vector< cadcells::Constraint > getConstraintsOfCovering (std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint >> &mDerivationToConstraint)`
- `std::optional< datastructures::SampledDerivationRef< PropSet > > constructDerivation (std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint >> &mDerivationToConstraint)`

### 0.15.289.1 Constructor & Destructor Documentation

**0.15.289.1.1 LevelWiseInformation()** [1/2] template<class Settings >  
`smrat::LevelWiseInformation< Settings >::LevelWiseInformation ()` [inline]

**0.15.289.1.2 LevelWiseInformation()** [2/2] template<class Settings >  
`smrat::LevelWiseInformation< Settings >::LevelWiseInformation (`  
`LevelWiseInformation< Settings > && other )` [inline]

### 0.15.289.2 Member Function Documentation

**0.15.289.2.1 addDerivation()** [1/2] template<class Settings >  
`void smrat::LevelWiseInformation< Settings >::addDerivation (`  
`const datastructures::SampledDerivationRef< PropSet > & derivation )` [inline]

```
0.15.289.2.2 addDerivation() [2/2] template<class Settings >
void smtrat::LevelWiseInformation< Settings >::addDerivation (
 datastructures::SampledDerivationRef< PropSet > && derivation) [inline]
```

```
0.15.289.2.3 clear() template<class Settings >
void smtrat::LevelWiseInformation< Settings >::clear () [inline]
```

```
0.15.289.2.4 computeCovering() template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::computeCovering () [inline]
```

```
0.15.289.2.5 constructDerivation() template<class Settings >
std::optional<datastructures::SampledDerivationRef<PropSet> > smtrat::LevelWiseInformation<
Settings >::constructDerivation (
 std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint
>> & mDerivationToConstraint) [inline]
```

```
0.15.289.2.6 getConstraintsOfCovering() template<class Settings >
std::vector<cadcells::Constraint> smtrat::LevelWiseInformation< Settings >::getConstraintsOfCovering (
 std::map< datastructures::SampledDerivationRef< PropSet >, std::vector< cadcells::Constraint
>> & mDerivationToConstraint) [inline]
```

```
0.15.289.2.7 getCoveringStatus() template<class Settings >
CoveringStatus smtrat::LevelWiseInformation< Settings >::getCoveringStatus () const [inline]
```

```
0.15.289.2.8 getDerivations() template<class Settings >
const std::set<datastructures::SampledDerivationRef<PropSet>, SampledDerivationRefCompare>&
smtrat::LevelWiseInformation< Settings >::getDerivations () const [inline]
```

```
0.15.289.2.9 getSampleOutside() template<class Settings >
const cadcells::RAN& smtrat::LevelWiseInformation< Settings >::getSampleOutside () const
[inline]
```

```
0.15.289.2.10 isFailedCovering() template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::isFailedCovering () const [inline]
```

```
0.15.289.2.11 isFullCovering() template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::isFullCovering () const [inline]
```

```
0.15.289.2.12 isPartialCovering() template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::isPartialCovering () const [inline]
```

```
0.15.289.2.13 isUnknownCovering() template<class Settings >
bool smtrat::LevelWiseInformation< Settings >::isUnknownCovering () const [inline]
```

---

**0.15.289.2.14 operator=()** template<class Settings >  
 LevelWiseInformation& smrat::LevelWiseInformation< Settings >::operator= (  
     LevelWiseInformation< Settings > && other ) [inline]

**0.15.289.2.15 removeConstraint()** template<class Settings >  
 void smrat::LevelWiseInformation< Settings >::removeConstraint (  
     const cadcells::Constraint & constraint,  
     const std::map< datastructures::SampledDerivationRef< PropSet >, std::vector<  
         cadcells::Constraint >> & derivationConstraints ) [inline]

**0.15.289.2.16 removeDerivation() [1/2]** template<class Settings >  
 void smrat::LevelWiseInformation< Settings >::removeDerivation (  
     const datastructures::SampledDerivationRef< PropSet > & derivation ) [inline]

**0.15.289.2.17 removeDerivation() [2/2]** template<class Settings >  
 void smrat::LevelWiseInformation< Settings >::removeDerivation (  
     const std::vector< datastructures::SampledDerivationRef< PropSet >> & derivations  
 ) [inline]

**0.15.289.2.18 setDerivations()** template<class Settings >  
 void smrat::LevelWiseInformation< Settings >::setDerivations (  
     std::vector< datastructures::SampledDerivationRef< PropSet >> && derivations )  
 [inline]

## 0.15.290 smrat::cad::LiftingTree< Settings > Class Template Reference

#include <LiftingTree.h>

### Public Types

- using Tree = carl::tree< Sample >
- using Iterator = Tree::iterator
- using FSC = FullSampleComparator< Iterator, Settings::fullSampleComparator >
- using SC = SampleComparator< Iterator, Settings::sampleComparator >
- using Constraints = CADConstraints< Settings::backtracking >

### Public Member Functions

- LiftingTree (const Constraints &c)
- LiftingTree (const LiftingTree &)=delete
- LiftingTree (LiftingTree &&)=delete
- LiftingTree & operator= (const LiftingTree &)=delete
- LiftingTree & operator= (LiftingTree &&)=delete
- const auto & getTree () const
- const auto & getLiftingQueue () const
- void reset (std::vector< carl::Variable > &&vars)
- bool hasFullSamples () const
- Iterator getNextFullSample ()
- void resetFullSamples ()
- bool hasNextSample () const
- Iterator getNextSample ()
- void removeNextSample ()

- void `restoreRemovedSamples ()`
- bool `liftSample (Iterator sample, const UPoly &p, std::size_t pid)`
- bool `addTrivialSample (Iterator sample)`
- Assignment `extractSampleMap (Iterator it) const`
- void `removedPolynomialsFromLevel (std::size_t level, const carl::Bitset &mask)`
- void `removedConstraint (const carl::Bitset &mask)`
- bool `is_consistent () const`
- std::string `printSample (Iterator sample) const`
- std::string `printFullSamples () const`

### 0.15.290.1 Member TypeDef Documentation

#### 0.15.290.1.1 Constraints template<typename Settings >

```
using smtrat::cad::LiftingTree< Settings >::Constraints = CADConstraints<Settings::backtracking>
```

#### 0.15.290.1.2 FSC template<typename Settings >

```
using smtrat::cad::LiftingTree< Settings >::FSC = FullSampleComparator<Iterator, Settings::fullSampleComparator>
```

#### 0.15.290.1.3 Iterator template<typename Settings >

```
using smtrat::cad::LiftingTree< Settings >::Iterator = Tree::iterator
```

#### 0.15.290.1.4 SC template<typename Settings >

```
using smtrat::cad::LiftingTree< Settings >::SC = SampleComparator<Iterator, Settings::sampleComparator>
```

#### 0.15.290.1.5 Tree template<typename Settings >

```
using smtrat::cad::LiftingTree< Settings >::Tree = carl::tree<Sample>
```

### 0.15.290.2 Constructor & Destructor Documentation

#### 0.15.290.2.1 LiftingTree() [1/3] template<typename Settings >

```
smtrat::cad::LiftingTree< Settings >::LiftingTree (
 const Constraints & c) [inline]
```

#### 0.15.290.2.2 LiftingTree() [2/3] template<typename Settings >

```
smtrat::cad::LiftingTree< Settings >::LiftingTree (
 const LiftingTree< Settings > &) [delete]
```

#### 0.15.290.2.3 LiftingTree() [3/3] template<typename Settings >

```
smtrat::cad::LiftingTree< Settings >::LiftingTree (
 LiftingTree< Settings > &&) [delete]
```

### 0.15.290.3 Member Function Documentation

**0.15.290.3.1 `addTrivialSample()`** template<typename Settings >  
bool smtrat::cad::LiftingTree< Settings >::addTrivialSample (  
    Iterator sample ) [inline]

**0.15.290.3.2 `extractSampleMap()`** template<typename Settings >  
Assignment smtrat::cad::LiftingTree< Settings >::extractSampleMap (  
    Iterator it ) const [inline]

**0.15.290.3.3 `getLiftingQueue()`** template<typename Settings >  
const auto& smtrat::cad::LiftingTree< Settings >::getLiftingQueue ( ) const [inline]

**0.15.290.3.4 `getNextFullSample()`** template<typename Settings >  
Iterator smtrat::cad::LiftingTree< Settings >::getNextFullSample ( ) [inline]

**0.15.290.3.5 `getNextSample()`** template<typename Settings >  
Iterator smtrat::cad::LiftingTree< Settings >::getNextSample ( ) [inline]

**0.15.290.3.6 `getTree()`** template<typename Settings >  
const auto& smtrat::cad::LiftingTree< Settings >::getTree ( ) const [inline]

**0.15.290.3.7 `hasFullSamples()`** template<typename Settings >  
bool smtrat::cad::LiftingTree< Settings >::hasFullSamples ( ) const [inline]

**0.15.290.3.8 `hasNextSample()`** template<typename Settings >  
bool smtrat::cad::LiftingTree< Settings >::hasNextSample ( ) const [inline]

**0.15.290.3.9 `is_consistent()`** template<typename Settings >  
bool smtrat::cad::LiftingTree< Settings >::is\_consistent ( ) const [inline]

**0.15.290.3.10 `liftSample()`** template<typename Settings >  
bool smtrat::cad::LiftingTree< Settings >::liftSample (  
    Iterator sample,  
    const UPoly & p,  
    std::size\_t pid ) [inline]

**0.15.290.3.11 `operator=() [1/2]`** template<typename Settings >  
LiftingTree& smtrat::cad::LiftingTree< Settings >::operator= (  
    const LiftingTree< Settings > & ) [delete]

**0.15.290.3.12 `operator=() [2/2]`** template<typename Settings >  
LiftingTree& smtrat::cad::LiftingTree< Settings >::operator= (  
    LiftingTree< Settings > && ) [delete]

**0.15.290.3.13 `printFullSamples()`** template<typename Settings >  
`std::string smrat::cad::LiftingTree< Settings >::printFullSamples ( ) const [inline]`

**0.15.290.3.14 `printSample()`** template<typename Settings >  
`std::string smrat::cad::LiftingTree< Settings >::printSample (`  
`Iterator sample ) const [inline]`

**0.15.290.3.15 `removedConstraint()`** template<typename Settings >  
`void smrat::cad::LiftingTree< Settings >::removedConstraint (`  
`const carl::Bitset & mask ) [inline]`

**0.15.290.3.16 `removedPolynomialsFromLevel()`** template<typename Settings >  
`void smrat::cad::LiftingTree< Settings >::removedPolynomialsFromLevel (`  
`std::size_t level,`  
`const carl::Bitset & mask ) [inline]`

**0.15.290.3.17 `removeNextSample()`** template<typename Settings >  
`void smrat::cad::LiftingTree< Settings >::removeNextSample ( ) [inline]`

**0.15.290.3.18 `reset()`** template<typename Settings >  
`void smrat::cad::LiftingTree< Settings >::reset (`  
`std::vector< carl::Variable > && vars ) [inline]`

**0.15.290.3.19 `resetFullSamples()`** template<typename Settings >  
`void smrat::cad::LiftingTree< Settings >::resetFullSamples ( ) [inline]`

**0.15.290.3.20 `restoreRemovedSamples()`** template<typename Settings >  
`void smrat::cad::LiftingTree< Settings >::restoreRemovedSamples ( ) [inline]`

## 0.15.291 smrat::resource::Limiter Class Reference

```
#include <ResourceLimitation.h>
```

### Public Member Functions

- void `initialize ()`
- void `reset ()`
- void `setMemout (std::size_t megabytes)`
- void `setTimeout (std::size_t seconds)`
- void `resetTimeout () const`
- void `setTimeoutHandler (std::function< void()> f)`
- std::function< void()> `timeoutHandler () const`

### 0.15.291.1 Member Function Documentation

**0.15.291.1.1 `initialize()`** void `smrat::resource::Limiter::initialize ( ) [inline]`

**0.15.291.1.2 `reset()`** `void smtrat::resource::Limiter::reset ( ) [inline]`

**0.15.291.1.3 `resetTimeout()`** `void smtrat::resource::Limiter::resetTimeout ( ) const [inline]`

**0.15.291.1.4 `setMemout()`** `void smtrat::resource::Limiter::setMemout ( std::size_t megabytes ) [inline]`

**0.15.291.1.5 `setTimeout()`** `void smtrat::resource::Limiter::setTimeout ( std::size_t seconds ) [inline]`

**0.15.291.1.6 `setTimeoutHandler()`** `void smtrat::resource::Limiter::setTimeoutHandler ( std::function< void()> f ) [inline]`

**0.15.291.1.7 `timeoutHandler()`** `std::function<void()> smtrat::resource::Limiter::timeoutHandler ( ) const [inline]`

## 0.15.292 `smtrat::ICPModule< Settings >::linearVariable` Struct Reference

```
#include <ICPModule.h>
```

### Data Fields

- `FormulaT constraint`
- `ConstraintT origin`

### 0.15.292.1 Field Documentation

**0.15.292.1.1 `constraint`** `template<class Settings > FormulaT smtrat::ICPModule< Settings >::linearVariable::constraint`

**0.15.292.1.2 `origin`** `template<class Settings > ConstraintT smtrat::ICPModule< Settings >::linearVariable::origin`

## 0.15.293 `smtrat::mcsat::ClauseChain::Link` Struct Reference

```
#include <ClauseChain.h>
```

### Public Member Functions

- `Link (const FormulaT &&clause, const FormulaT &&impliedTseitinLiteral)`
- `Link (const FormulaT &&clause)`
- `bool isPropagating () const`
- `bool isConflicting () const`
- `bool isOptional () const`
- `const FormulaT & clause () const`
- `const FormulaT & impliedTseitinLiteral () const`

**Friends**

- std::ostream & `operator<<` (std::ostream &stream, const `Link` &link)

**0.15.293.1 Constructor & Destructor Documentation**

**0.15.293.1.1 `Link()` [1/2]** smtrat::mcsat::ClauseChain::Link::Link (const `FormulaT` && clause, const `FormulaT` && impliedTseitinLiteral ) [inline]

**0.15.293.1.2 `Link()` [2/2]** smtrat::mcsat::ClauseChain::Link::Link (const `FormulaT` && clause ) [inline]

**0.15.293.2 Member Function Documentation**

**0.15.293.2.1 `clause()`** const `FormulaT&` smtrat::mcsat::ClauseChain::Link::clause ( ) const [inline]

**0.15.293.2.2 `impliedTseitinLiteral()`** const `FormulaT&` smtrat::mcsat::ClauseChain::Link::impliedTseitinLiteral ( ) const [inline]

**0.15.293.2.3 `isConflicting()`** bool smtrat::mcsat::ClauseChain::Link::isConflicting ( ) const [inline]

**0.15.293.2.4 `isOptional()`** bool smtrat::mcsat::ClauseChain::Link::isOptional ( ) const [inline]

**0.15.293.2.5 `isPropagating()`** bool smtrat::mcsat::ClauseChain::Link::isPropagating ( ) const [inline]

**0.15.293.3 Friends And Related Function Documentation**

**0.15.293.3.1 `operator<<`** std::ostream& operator<< (std::ostream & stream, const `Link` & link ) [friend]

**0.15.294 Minisat::Lit Struct Reference**

```
#include <SolverTypes.h>
```

**Public Member Functions**

- bool `operator==(Lit p)` const
- bool `operator!=(Lit p)` const
- bool `operator<(Lit p)` const

**Data Fields**

- int `x`

## Friends

- `Lit mkLit (Var var, bool sign)`

### 0.15.294.1 Member Function Documentation

**0.15.294.1.1 `operator"!=()`** `bool Minisat::Lit::operator!= (`  
`Lit p ) const [inline]`

**0.15.294.1.2 `operator<()`** `bool Minisat::Lit::operator< (`  
`Lit p ) const [inline]`

**0.15.294.1.3 `operator==()`** `bool Minisat::Lit::operator== (`  
`Lit p ) const [inline]`

### 0.15.294.2 Friends And Related Function Documentation

**0.15.294.2.1 `mkLit`** `Lit mkLit (`  
`Var var,`  
`bool sign = false ) [friend]`

### 0.15.294.3 Field Documentation

**0.15.294.3.1 `x`** `int Minisat::Lit::x`

## 0.15.295 `benchmax::LocalBackend` Class Reference

This backend simply runs files sequentially on the local machine.

```
#include <LocalBackend.h>
```

### Public Member Functions

- `bool suspendable () const`
- `void process_results (const Jobs &jobs, bool check_finished)`
- `void addResult (const Tool *tool, const fs::path &file, BenchmarkResult &&result)`  
*Add a result.*
- `void run (const Jobs &jobs, bool wait_for_termination)`  
*Run the list of tools against the list of benchmarks.*

### Protected Member Functions

- `virtual void execute (const Tool *tool, const fs::path &file)`  
*Execute the tool on the file manually.*
- `virtual void startTool (const Tool *)`  
*Hook for every tool at the beginning.*
- `virtual void finalize ()`  
*Hook to allow for asynchronous backends to wait for jobs to terminate.*
- `void madeProgress (std::size_t files=1)`

*Can be called to give information about the current progress, if available.*

- virtual bool `collect_results` (const `Jobs` &, bool)
- void `sanitize_results` (const `Jobs` &`jobs`) const
- void `write_results` (const `Jobs` &`jobs`) const

## Protected Attributes

- std::size\_t `mExpectedJobs`  
*Number of jobs that should be run.*
- std::atomic< std::size\_t > `mFinishedJobs`  
*Number of jobs that are finished.*
- std::atomic< std::size\_t > `mLastPercent`  
*Percentage of finished jobs when `madeProgress()` was last called.*

### 0.15.295.1 Detailed Description

This backend simply runs files sequentially on the local machine.

### 0.15.295.2 Member Function Documentation

**0.15.295.2.1 `addResult()`** void `benchmax::Backend::addResult` (  
    const `Tool` \* `tool`,  
    const `fs::path` & `file`,  
    `BenchmarkResult` && `result` ) [inline], [inherited]

Add a result.

**0.15.295.2.2 `collect_results()`** virtual bool `benchmax::Backend::collect_results` (  
    const `Jobs` & ,  
    bool ) [inline], [protected], [virtual], [inherited]

**0.15.295.2.3 `execute()`** virtual void `benchmax::LocalBackend::execute` (  
    const `Tool` \* `tool`,  
    const `fs::path` & `file` ) [inline], [protected], [virtual]  
Execute the tool on the file manually.  
Reimplemented from `benchmax::Backend`.

**0.15.295.2.4 `finalize()`** virtual void `benchmax::Backend::finalize` () [inline], [protected], [virtual], [inherited]  
Hook to allow for asynchronous backends to wait for jobs to terminate.

**0.15.295.2.5 `madeProgress()`** void `benchmax::Backend::madeProgress` (  
    std::size\_t `files` = 1 ) [inline], [protected], [inherited]  
Can be called to give information about the current progress, if available.

**0.15.295.2.6 `process_results()`** void `benchmax::Backend::process_results` (  
    const `Jobs` & `jobs`,  
    bool `check_finished` ) [inline], [inherited]

**0.15.295.2.7 run()** void benchmax::Backend::run ( const [Jobs](#) & jobs, bool [wait\\_for\\_termination](#) ) [inline], [inherited]

Run the list of tools against the list of benchmarks.

**0.15.295.2.8 sanitize\_results()** void benchmax::Backend::sanitize\_results ( const [Jobs](#) & jobs ) const [inline], [protected], [inherited]

**0.15.295.2.9 startTool()** virtual void benchmax::Backend::startTool ( const [Tool](#) \* ) [inline], [protected], [virtual], [inherited]

Hook for every tool at the beginning.

Can be used to upload the tool to some remote system.

**0.15.295.2.10 suspendable()** bool benchmax::Backend::suspendable () const [inline], [inherited]

**0.15.295.2.11 write\_results()** void benchmax::Backend::write\_results ( const [Jobs](#) & jobs ) const [inline], [protected], [inherited]

### 0.15.295.3 Field Documentation

**0.15.295.3.1 mExpectedJobs** std::size\_t benchmax::Backend::mExpectedJobs [protected], [inherited]  
Number of jobs that should be run.

**0.15.295.3.2 mFinishedJobs** std::atomic<std::size\_t> benchmax::Backend::mFinishedJobs [protected], [inherited]  
Number of jobs that are finished.

**0.15.295.3.3 mLastPercent** std::atomic<std::size\_t> benchmax::Backend::mLastPercent [protected], [inherited]  
Percentage of finished jobs when [madeProgress\(\)](#) was last called.

## 0.15.296 smrat::LOG Class Reference

#include <LOG.h>

### Public Member Functions

- bool [isDebugEnabled](#) ()

### Data Fields

- bool [debugEnabled](#) = false

### Friends

- class [carl::Singleton< LOG >](#)

### 0.15.296.1 Member Function Documentation

**0.15.296.1.1 `isDebugEnabled()`** `bool smtrat::LOG::isDebugEnabled ()` [inline]

### 0.15.296.2 Friends And Related Function Documentation

**0.15.296.2.1 `carl::Singleton< LOG >`** `friend class carl::Singleton< LOG >` [friend]

### 0.15.296.3 Field Documentation

**0.15.296.3.1 `debugEnabled`** `bool smtrat::LOG::debugEnabled = false`

## 0.15.297 `smtrat::parser::LogicParser` Struct Reference

```
#include <Script.h>
```

### Public Member Functions

- [LogicParser \(\)](#)

### 0.15.297.1 Constructor & Destructor Documentation

**0.15.297.1.1 `LogicParser()`** `smtrat::parser::LogicParser::LogicParser ()` [inline]

## 0.15.298 `smtrat::LongFormulaEncoder` Class Reference

```
#include <LongFormulaEncoder.h>
```

### Public Member Functions

- [LongFormulaEncoder \(\)](#)
- `bool canEncode (const ConstraintT &constraint)`
- [Rational encodingSize \(const ConstraintT &constraint\)](#)
- `std::string name ()`
- `std::optional< FormulaT > encode (const ConstraintT &constraint)`

*Encodes an arbitrary constraint.*

### Data Fields

- `std::size_t problem_size`

### Protected Member Functions

- `std::optional< FormulaT > doEncode (const ConstraintT &constraint)`
- `FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)`

### 0.15.298.1 Constructor & Destructor Documentation

**0.15.298.1.1 `LongFormulaEncoder()`** `smtrat::LongFormulaEncoder::LongFormulaEncoder ()` [inline]

### 0.15.298.2 Member Function Documentation

**0.15.298.2.1 canEncode()** `bool smtrat::LongFormulaEncoder::canEncode (`  
    `const ConstraintT & constraint ) [virtual]`  
Implements [smtrat::PseudoBoolEncoder](#).

**0.15.298.2.2 doEncode()** `std::optional< FormulaT > smtrat::LongFormulaEncoder::doEncode (`  
    `const ConstraintT & constraint ) [protected], [virtual]`  
Implements [smtrat::PseudoBoolEncoder](#).

**0.15.298.2.3 encode()** `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (`  
    `const ConstraintT & constraint ) [inherited]`  
Encodes an arbitrary constraint.

Returns

encoded formula

**0.15.298.2.4 encodingSize()** `Rational smtrat::LongFormulaEncoder::encodingSize (`  
    `const ConstraintT & constraint ) [virtual]`  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

**0.15.298.2.5 generateVarChain()** `FormulaT smtrat::PseudoBoolEncoder::generateVarChain (`  
    `const std::set< carl::Variable > & vars,`  
    `carl::FormulaType type ) [protected], [inherited]`

**0.15.298.2.6 name()** `std::string smtrat::LongFormulaEncoder::name ( ) [inline], [virtual]`  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

### 0.15.298.3 Field Documentation

**0.15.298.3.1 problem\_size** `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

## 0.15.299 smtrat::LRAmodule< Settings > Class Template Reference

A module which performs the Simplex method on the linear part of it's received formula.

```
#include <LRAmodule.h>
```

### Data Structures

- struct [Context](#)

*Stores a formula, being part of the received formula of this module, and the position of this formula in the passed formula.*

## Public Types

- **typedef Settings::BoundType LRABoundType**  
*Number type of the underlying value of a bound of a variable within the `LRAModule`.*
- **typedef Settings::EntryType LRAEntryType**  
*Type of an entry within the tableau.*
- **typedef Ira::Bound< LRABoundType, LRAEntryType > LRABound**  
*Type of a bound of a variable within the `LRAModule`.*
- **typedef Ira::Variable< LRABoundType, LRAEntryType > LRAVariable**  
*A variable of the `LRAModule`, being either a original variable or a slack variable representing a linear polynomial.*
- **typedef Ira::Value< LRABoundType > LRAValue**  
*The type of the assignment of a variable maintained by the `LRAModule`. It consists of a tuple of two value of the bound type.*
- **typedef Settings SettingsType**
- **typedef carl::FastMap< carl::Variable, LRAVariable \* > VarVariableMap**  
*Maps an original variable to it's corresponding `LRAModule` variable.*
- **typedef carl::FastPointerMap< typename Poly::PolyType, LRAVariable \* > ExVariableMap**  
*Maps a linear polynomial to it's corresponding `LRAModule` variable.*
- **typedef carl::FastMap< FormulaT, std::vector< const LRABound \* > \* > ConstraintBoundsMap**  
*Maps constraint to the bounds it represents (e.g., equations represent two bounds)*
- **typedef carl::FastMap< FormulaT, Context > ConstraintContextMap**  
*Stores the position of a received formula in the passed formula.*
- **typedef Ira::Tableau< typename Settings::Tableau\_settings, LRABoundType, LRAEntryType > LRATableau**  
*The tableau which is the main data structure maintained by the `LRAModule` responsible for the pivoting steps.*
- **enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }**

## Public Member Functions

- **std::string moduleName () const**
- **LRAModule (const ModuleInput \* \_formula, Conditionals & \_conditionals, Manager \* \_manager=NULL)**  
*Constructs a `LRAModule`.*
- **virtual ~LRAModule ()**  
*Destructs this `LRAModule`.*
- **bool informCore (const FormulaT & \_constraint)**  
*Informs this module about the existence of the given constraint, which means that it could be added in future.*
- **void deinformCore (const FormulaT & \_constraint)**  
*The inverse of informing about a constraint.*
- **void init ()**  
*Initializes the tableau according to all linear constraints, of which this module has been informed.*
- **bool addCore (ModuleInput::const\_iterator \_subformula)**  
*The module has to take the given sub-formula of the received formula into account.*
- **void removeCore (ModuleInput::const\_iterator \_subformula)**  
*Removes everything related to the given sub-formula of the received formula.*
- **Answer checkCore ()**  
*Checks the received formula for consistency.*
- **Answer checkCore\_old ()**
- **Answer processResult (Answer \_result)**
- **void updateModel () const**  
*Updates the model, if the solver has detected the consistency of the received formula, beforehand.*
- **unsigned currentlySatisfied (const FormulaT &) const**
- **const RationalAssignment & getRationalModel () const**  
*Gives a rational model if the received formula is satisfiable.*

- `Answer optimize (Answer _result)`
- `Answer checkNotEqualConstraints (Answer _result)`
- `void processLearnedBounds ()`
- `void createInfeasibleSubsets (Ira::EntryID _tableauEntry)`
- `EvalRationalIntervalMap getVariableBounds () const`

*Returns the bounds of the variables as intervals.*
- `void printLinearConstraints (std::ostream &_out=std::cout, const std::string _init="") const`

*Prints all linear constraints.*
- `void printNonlinearConstraints (std::ostream &_out=std::cout, const std::string _init="") const`

*Prints all non-linear constraints.*
- `void printConstraintToBound (std::ostream &_out=std::cout, const std::string _init="") const`

*Prints the mapping of constraints to their corresponding bounds.*
- `void printBoundCandidatesToPass (std::ostream &_out=std::cout, const std::string _init="") const`

*Prints the strictest bounds, which have to be passed the backend in case.*
- `void printRationalModel (std::ostream &_out=std::cout, const std::string _init="") const`

*Prints the current rational assignment.*
- `void printTableau (std::ostream &_out=std::cout, const std::string _init="") const`

*Prints the current tableau.*
- `void printVariables (std::ostream &_out=std::cout, const std::string _init="") const`

*Prints all Ira variables and their assignments.*
- `const VarVariableMap & originalVariables () const`
- `const ExVariableMap & slackVariables () const`
- `const LRAVariable * getSlackVariable (const FormulaT &_constraint) const`
- `bool inform (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- `void deinform (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- `bool add (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- `virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`

*Checks the received formula for consistency.*
- `virtual void remove (ModuleInput::const_iterator _subformula)`

*Removes everything related to the given sub-formula of the received formula.*
- `virtual void updateAllModels ()`

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`

*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`

- const std::vector< Model > & **allModels** () const
- const std::vector< FormulaSetT > & **infeasibleSubsets** () const
- const std::vector< Module \* > & **usedBackends** () const
- const carl::FastSet< FormulaT > & **constraintsToInform** () const
- const carl::FastSet< FormulaT > & **informedConstraints** () const
- void **addLemma** (const FormulaT &\_lemma, const LemmaType &\_lt=LemmaType::NORMAL, const FormulaT &\_preferredFormula=FormulaT(carl::FormulaType::TRUE))
 

*Stores a lemma being a valid formula.*
- bool **hasLemmas** ()
 

*Checks whether this module or any of its backends provides any lemmas.*
- void **clearLemmas** ()
 

*Deletes all yet found lemmas.*
- const std::vector< Lemma > & **lemmas** () const
- ModuleInput::const\_iterator **firstUncheckedReceivedSubformula** () const
- ModuleInput::const\_iterator **firstSubformulaToPass** () const
- void **receivedFormulaChecked** ()
 

*Notifies that the received formulas has been checked.*
- const smrat::Conditionals & **answerFound** () const
- bool **isPreprocessor** () const
- carl::Variable **objective** () const
- bool **is\_minimizing** () const
- void **excludeNotReceivedVariablesFromModel** () const
 

*Excludes all variables from the current model, which do not occur in the received formula.*
- void **updateLemmas** ()
 

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void **collectTheoryPropagations** ()
- void **collectOrigins** (const FormulaT &\_formula, FormulasT &\_origins) const
 

*Collects the formulas in the given formula, which are part of the received formula.*
- void **collectOrigins** (const FormulaT &\_formula, FormulaSetT &\_origins) const
- bool **hasValidInfeasibleSubset** () const
- void **checkInfSubsetForMinimality** (std::vector< FormulaSetT >::const\_iterator \_insubset, const std::string &filename="smaller\_muses", unsigned \_maxSizeDifference=1) const
 

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, FormulaT > **getReceivedFormulaSimplified** ()
- void **print** (const std::string &\_initiation="\*\*\*") const
 

*Prints everything relevant of the solver.*
- void **printReceivedFormula** (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of the received formula.*
- void **printPassedFormula** (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of passed formula.*
- void **printInfeasibleSubsets** (const std::string &\_initiation="\*\*\*") const
 

*Prints the infeasible subsets.*
- void **printModel** (std::ostream &\_out=std::cout) const
 

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void **printAllModels** (std::ostream &\_out=std::cout)
 

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void **freeSplittingVariable** (const FormulaT &\_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin` ()
- `ModuleInput::iterator passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` `_formula`, const `FormulaT` & `_origin`)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` `_formula`) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` & `_formula`, `FormulasT` & `_origins`) const
- void `getOrigins` (const `FormulaT` & `_formula`, `FormulaSetT` & `_origins`) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` `_formula`, const `FormulaT` & `_origin`)  
*Removes the origin of the given formula from the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` `_formula`, const std::shared\_ptr< std::vector< `FormulaT` >> & `_origins`)  
*Removes all origins of the given formula from the passed formula.*
- void `informBackends` (const `FormulaT` & `_constraint`)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` & `_constraint`)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` `_subformula`)  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` & `_origin`) const  
*Checks if the given formula is an origin of the received formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` & `_formula`)  
*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` & `_formula`, const std::shared\_ptr< std::vector< `FormulaT` >> & `_origins`)  
*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` & `_formula`, const `FormulaT` & `_origin`)  
*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()  
*Stores the trivial infeasible subset being the set of received formulas.*

- void `receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin (const std::vector< FormulaT > &_origins) const`
- void `getInfeasibleSubsets ()`

*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets (const Module &_backend) const`

*Get the infeasible subsets the given backend provides.*
- const `Model & backendsModel () const`

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel () const`
- void `getBackendsAllModels () const`

*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`
- std::vector< `FormulaT` > `merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`

*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin (const std::vector< FormulaT > &origins) const`
- bool `probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_← branchingValue) const`

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &← _premise=std::vector< FormulaT >())`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())`
- void `splitUnequalConstraint (const FormulaT &)`

*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel () const`
- void `addInformationRelevantFormula (const FormulaT &formula)`

*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas ()`

*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel (LemmaLevel level)`

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint (const Model &_modelA, const Model &_modelB)`

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< [FormulaSetT](#) > mInfeasibleSubsets
  - Stores the infeasible subsets.*
- Manager \*const mpManager
  - A reference to the manager.*
- Model mModel
  - Stores the assignment of the current satisfiable result, if existent.*
- std::vector< [Model](#) > mAllModels
  - Stores all satisfying assignments.*
- bool mModelComputed
  - True, if the model has already been computed.*
- bool mFinalCheck
  - true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool mFullCheck
  - false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable mObjectiveVariable
  - Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< [TheoryPropagation](#) > mTheoryPropagations
- std::atomic< [Answer](#) > mSolverState
  - States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* mBackendsFoundAnswer
  - This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals mFoundAnswer
  - Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< [Module](#) \* > mUsedBackends
  - The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< [Module](#) \* > mAllBackends
  - The backends of this module which have been used.*
- std::vector< [Lemma](#) > mLemmas
  - Stores the lemmas being valid formulas this module or its backends made.*
- ModuleInput::iterator mFirstSubformulaToPass
  - Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< [FormulaT](#) > mConstraintsToInform
  - Stores the constraints which the backends must be informed about.*
- carl::FastSet< [FormulaT](#) > mInformedConstraints
  - Stores the position of the first constraint of which no backend has been informed about.*
- ModuleInput::const\_iterator mFirstUncheckedReceivedSubformula
  - Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned mSmallerMusesCheckCounter
  - Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > mVariableCounters
  - Maps variables to the number of their occurrences.*

### 0.15.299.1 Detailed Description

```
template<class Settings>
class smrat::LRAModule< Settings >
```

A module which performs the Simplex method on the linear part of it's received formula.

### 0.15.299.2 Member Typedef Documentation

**0.15.299.2.1 ConstraintBoundsMap** template<class Settings >  
 typedef carl::FastMap<FormulaT, std::vector<const LRABound\*>> smtrat::LRAModule< Settings >::ConstraintBoundsMap  
 Maps constraint to the bounds it represents (e.g., equations represent two bounds)

**0.15.299.2.2 ConstraintContextMap** template<class Settings >  
 typedef carl::FastMap<FormulaT, Context> smtrat::LRAModule< Settings >::ConstraintContextMap  
 Stores the position of a received formula in the passed formula.

**0.15.299.2.3 ExVariableMap** template<class Settings >  
 typedef carl::FastPointerMap<typename Poly::PolyType, LRAVariable\*> smtrat::LRAModule< Settings >::ExVariableMap  
 Maps a linear polynomial to its corresponding **LRAModule** variable.

**0.15.299.2.4 LRABound** template<class Settings >  
 typedef lra::Bound<LRABoundType, LRAEntryType> smtrat::LRAModule< Settings >::LRABound  
 Type of a bound of a variable within the **LRAModule**.

**0.15.299.2.5 LRABoundType** template<class Settings >  
 typedef Settings::BoundType smtrat::LRAModule< Settings >::LRABoundType  
 Number type of the underlying value of a bound of a variable within the **LRAModule**.

**0.15.299.2.6 LRAEntryType** template<class Settings >  
 typedef Settings::EntryType smtrat::LRAModule< Settings >::LRAEntryType  
 Type of an entry within the tableau.

**0.15.299.2.7 LRATableau** template<class Settings >  
 typedef lra::Tableau<typename Settings::Tableau\_settings, LRABoundType, LRAEntryType> smtrat::LRAModule< Settings >::LRATableau  
 The tableau which is the main data structure maintained by the **LRAModule** responsible for the pivoting steps.

**0.15.299.2.8 LRAValue** template<class Settings >  
 typedef lra::Value<LRABoundType> smtrat::LRAModule< Settings >::LRAValue  
 The type of the assignment of a variable maintained by the **LRAModule**. It consists of a tuple of two values of the bound type.

**0.15.299.2.9 LRAVariable** template<class Settings >  
 typedef lra::Variable<LRABoundType, LRAEntryType> smtrat::LRAModule< Settings >::LRAVariable  
 A variable of the **LRAModule**, being either a original variable or a slack variable representing a linear polynomial.

**0.15.299.2.10 SettingsType** template<class Settings >  
 typedef Settings smtrat::LRAModule< Settings >::SettingsType

**0.15.299.2.11 VarVariableMap** template<class Settings >  
typedef carl::FastMap<carl::Variable, LRAVariable\*> smtrat::LRAModule< Settings >::VarVariableMap  
Maps an original variable to its corresponding [LRAModule](#) variable.

### 0.15.299.3 Member Enumeration Documentation

**0.15.299.3.1 LemmaType** enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.299.4 Constructor & Destructor Documentation

**0.15.299.4.1 LRAModule()** template<class Settings >  
smtrat::LRAModule< Settings >::LRAModule (   
    const ModuleInput \* \_formula,  
    Conditionals & \_conditionals,  
    Manager \* \_manager = NULL )

Constructs a [LRAModule](#).

Parameters

|               |                                                                                             |
|---------------|---------------------------------------------------------------------------------------------|
| _type         | The type of this module being <a href="#">LRAModule</a> .                                   |
| _formula      | The formula passed to this module, called received formula.                                 |
| _settings     | [Not yet used.]                                                                             |
| _conditionals | Vector of Booleans: If any of them is true, we have to terminate a running check procedure. |
| _manager      | A reference to the manager of the solver using this module.                                 |

**0.15.299.4.2 ~LRAModule()** template<class Settings >  
virtual smtrat::LRAModule< Settings >::~LRAModule ( ) [virtual]  
Destructs this [LRAModule](#).

### 0.15.299.5 Member Function Documentation

**0.15.299.5.1 add()** bool smtrat::Module::add (   
    ModuleInput::const\_iterator \_subformula ) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.299.5.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`

```
const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.299.5.3 addCore()** `template<class Settings >`

```
bool smtrat::LRAbstractModule< Settings >::addCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.299.5.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula (`

```
const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.299.5.5 addLemma()** `void smtrat::Module::addLemma (`

```
const FormulaT & _lemma,
```

```
const LemmaType & _lt = LemmaType::NORMAL,
```

```
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
```

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.299.5.6 addOrigin()** `void smtrat::Module::addOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.299.5.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.299.5.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.299.5.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.299.5.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>**

```
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.299.5.11 allModels() const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]****Returns**

All satisfying assignments, if existent.

**0.15.299.5.12 anAnswerFound() bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.299.5.13 answerFound() const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.299.5.14 backendsModel() const Model & smtrat::Module::backendsModel () const [protected], [inherited]**

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.299.5.15** **branchAt()** [1/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.299.5.16** **branchAt()** [2/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.299.5.17** **branchAt()** [3/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.299.5.18** **branchAt()** [4/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.299.5.19 check() Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

```
0.15.299.5.20 checkCore() template<class Settings >
Answer smtrat::LRAbstract< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.299.5.21 checkCore_old() template<class Settings >
Answer smtrat::LRAbstract< Settings >::checkCore_old ()
```

```
0.15.299.5.22 checkInfSubsetForMinimality() void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.299.5.23 checkModel()** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.299.5.24 checkNotEqualConstraints()** `template<class Settings >`

```
Answer smtrat::LRAModule< Settings >::checkNotEqualConstraints (
 Answer _result)
```

**0.15.299.5.25 cleanModel()** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.299.5.26 clearLemmas()** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

**0.15.299.5.27 clearModel()** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.299.5.28 clearModels()** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.299.5.29 clearPassedFormula()** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.299.5.30 collectOrigins() [1/2]** `void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins ) const [inherited]`

**0.15.299.5.31 collectOrigins() [2/2]** `void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.299.5.32 `collectTheoryPropagations()`** void smtrat::Module::collectTheoryPropagations ( )  
[inherited]

**0.15.299.5.33 `constraintsToInform()`** const carl::FastSet<[FormulaT](#)>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

#### Returns

The constraints which the backends must be informed about.

**0.15.299.5.34 `createInfeasibleSubsets()`** template<class Settings >  
void smtrat::LRAbstractModule< Settings >::createInfeasibleSubsets (   
 lra::EntryID \_tableauEntry )

**0.15.299.5.35 `currentlySatisfied()`** template<class Settings >  
unsigned smtrat::LRAbstractModule< Settings >::currentlySatisfied (   
 const [FormulaT](#) & ) const [virtual]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise

Reimplemented from [smtrat::Module](#).

**0.15.299.5.36 `currentlySatisfiedByBackend()`** unsigned smtrat::Module::currentlySatisfiedByBackend (   
 const [FormulaT](#) & \_formula ) const [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.299.5.37 `deinform()`** void smtrat::Module::deinform (   
 const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.299.5.38 `deinformCore()`** template<class Settings >  
void smtrat::LRAbstractModule< Settings >::deinformCore (   
 const [FormulaT](#) & \_constraint ) [virtual]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.299.5.39 `determine_smallest_origin()`** `size_t smrat::Module::determine_smallest_origin (const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.299.5.40 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smrat::Module::eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.299.5.41 `excludeNotReceivedVariablesFromModel()`** `void smrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.299.5.42 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smrat::Module::findBestOrigin (const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

Returns

**0.15.299.5.43 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.299.5.44 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.299.5.45 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.299.5.46 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.299.5.47 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.299.5.48 getBackendsModel()** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.299.5.49 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.299.5.50 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.299.5.51 getInformationRelevantFormulas()** const std::set< [FormulaT](#) > & smtrat::Module::getInformationRelevantFormulas () [protected], [inherited]  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.299.5.52 getModelEqualities()** std::list< std::vector< [carl::Variable](#) > > smtrat::Module::getModelEqualities () const [virtual], [inherited]  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.299.5.53 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const [FormulaT](#) & *\_formula*, [FormulaSetT](#) & *\_origins* ) const [inline], [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_formula</i> |  |
| <i>_origins</i> |  |

**0.15.299.5.54 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const [FormulaT](#) & *\_formula*, [FormulasT](#) & *\_origins* ) const [inline], [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_formula</i> |  |
| <i>_origins</i> |  |

**0.15.299.5.55 getOrigins() [3/3]** const [FormulaT](#)& smtrat::Module::getOrigins ( [ModuleInput::const\\_iterator](#) *\_formula* ) const [inline], [protected], [inherited]  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.299.5.56 `getRationalModel()`** template<class Settings >  
 const RationalAssignment& smtrat::LRAutoModule< Settings >::getRationalModel ( ) const  
 Gives a rational model if the received formula is satisfiable.  
 Note, that it is calculated from scratch every time you call this method.

**Returns**

The rational model.

**0.15.299.5.57 `getReceivedFormulaSimplified()`** std::pair< bool, FormulaT > smtrat::Module::get<= ReceivedFormulaSimplified ( ) [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.299.5.58 `getSlackVariable()`** template<class Settings >  
 const LRAVariable\* smtrat::LRAutoModule< Settings >::getSlackVariable (   
 const FormulaT & \_constraint ) const [inline]

**Parameters**

|                          |                                               |
|--------------------------|-----------------------------------------------|
| <code>_constraint</code> | The constraint to get the slack variable for. |
|--------------------------|-----------------------------------------------|

**Returns**

The slack variable constructed for the linear polynomial without the constant part in the given constraint.

**0.15.299.5.59 `getVariableBounds()`** template<class Settings >  
 EvalRationalIntervalMap smtrat::LRAutoModule< Settings >::getVariableBounds ( ) const  
 Returns the bounds of the variables as intervals.

**Returns**

The bounds of the variables as intervals.

**0.15.299.5.60 `hasLemmas()`** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]  
 Checks whether this module or any of its backends provides any lemmas.

**0.15.299.5.61 `hasValidInfeasibleSubset()`** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.299.5.62** `id()` `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.299.5.63** `infeasibleSubsets()` `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.299.5.64** `inform()` `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.299.5.65** `informBackends()` `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.299.5.66** `informCore()` `template<class Settings> bool smtrat::LRAbstractModule<Settings>::informCore (const FormulaT & _constraint) [virtual]`

Informs this module about the existence of the given constraint, which means that it could be added in future.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if the it can be determined that the constraint itself is conflicting; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.299.5.67 informedConstraints()** const carl::FastSet<[FormulaT](#)>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]

#### Returns

The position of the first constraint of which no backend has been informed about.

**0.15.299.5.68 init()** template<class Settings >  
void [smtrat::LRAbstractModule](#)< Settings >::init ( ) [virtual]

Initializes the tableau according to all linear constraints, of which this module has been informed.  
Reimplemented from [smtrat::Module](#).

**0.15.299.5.69 is\_minimizing()** bool smtrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.299.5.70 isLemmaLevel()** bool smtrat::Module::isLemmaLevel ( [LemmaLevel](#) level ) [protected], [inherited]

Checks if current lemma level is greater or equal to given level.

#### Parameters

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.299.5.71 isPreprocessor()** bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]

#### Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.299.5.72 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

#### Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.299.5.73 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & *\_vecSetA*, const std::vector< [FormulaT](#) > & *\_vecSetB* ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

#### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.299.5.74 model()** const [Model](#)& smtrat::Module::model () const [inline], [inherited]**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.299.5.75 modelsDisjoint()** bool smtrat::Module::modelsDisjoint (

```
const Model & _modelA,
const Model & _modelB) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

**Parameters**

|         |                                |
|---------|--------------------------------|
| _modelA | The first model to check for.  |
| _modelB | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.299.5.76 moduleName()** template<class Settings >

```
std::string smtrat::LRAModule< Settings >::moduleName () const [inline], [virtual]
```

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.299.5.77 objective()** [carl::Variable](#) smtrat::Module::objective () const [inline], [inherited]**0.15.299.5.78 optimize()** template<class Settings >

```
Answer smtrat::LRAModule< Settings >::optimize (
 Answer _result)
```

**0.15.299.5.79 originalVariables()** template<class Settings >

```
const VarVariableMap& smtrat::LRAModule< Settings >::originalVariables () const [inline]
```

**Returns**

A constant reference to the original variables.

**0.15.299.5.80 originInReceivedFormula()** bool smtrat::Module::originInReceivedFormula (

```
const FormulaT & _origin) const [protected], [inherited]
```

**0.15.299.5.81 passedFormulaBegin()** `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )` [inline], [protected], [inherited]

#### Returns

An iterator to the end of the passed formula. TODO: disable this method

**0.15.299.5.82 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )` [inline], [protected], [inherited]

#### Returns

An iterator to the end of the passed formula.

**0.15.299.5.83 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const` [inline], [inherited]

#### Returns

A pointer to the formula passed to the backends of this module.

**0.15.299.5.84 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const` [inline], [inherited]

#### Returns

A pointer to the formula passed to this module.

**0.15.299.5.85 print()** `void smtrat::Module::print ( const std::string & _initiation = "***" ) const`

Prints everything relevant of the solver.

#### Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.299.5.86 printAllModels()** `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

#### Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.299.5.87 printBoundCandidatesToPass()** `template<class Settings > void smtrat::LRAModule< Settings >::printBoundCandidatesToPass ( std::ostream & _out = std::cout, const std::string _init = "" ) const`

Prints the strictest bounds, which have to be passed the backend in case.

#### Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <code>_out</code>  | The output stream to print on.       |
| <code>_init</code> | The beginning of each line to print. |

```
0.15.299.5.88 printConstraintToBound() template<class Settings >
void smtrat::LRAModule< Settings >::printConstraintToBound (
 std::ostream & _out = std::cout,
 const std::string _init = "") const
```

Prints the mapping of constraints to their corresponding bounds.

#### Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <code>_out</code>  | The output stream to print on.       |
| <code>_init</code> | The beginning of each line to print. |

```
0.15.299.5.89 printInfeasibleSubsets() void smtrat::Module::printInfeasibleSubsets (
 const std::string & _initiation = "***") const [inherited]
```

Prints the infeasible subsets.

#### Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

```
0.15.299.5.90 printLinearConstraints() template<class Settings >
void smtrat::LRAModule< Settings >::printLinearConstraints (
 std::ostream & _out = std::cout,
 const std::string _init = "") const
```

Prints all linear constraints.

#### Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <code>_out</code>  | The output stream to print on.       |
| <code>_init</code> | The beginning of each line to print. |

```
0.15.299.5.91 printModel() void smtrat::Module::printModel (
 std::ostream & _out = std::cout) const [inherited]
```

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

#### Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

```
0.15.299.5.92 printNonlinearConstraints() template<class Settings >
void smtrat::LRAModule< Settings >::printNonlinearConstraints (
 std::ostream & _out = std::cout,
 const std::string _init = "") const
Prints all non-linear constraints.
```

**Parameters**

|              |                                      |
|--------------|--------------------------------------|
| <i>_out</i>  | The output stream to print on.       |
| <i>_init</i> | The beginning of each line to print. |

```
0.15.299.5.93 printPassedFormula() void smtrat::Module::printPassedFormula (
 const std::string & _initiation = "***") const [inherited]
Prints the vector of passed formula.
```

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.299.5.94 printRationalModel() template<class Settings >
void smtrat::LRAModule< Settings >::printRationalModel (
 std::ostream & _out = std::cout,
 const std::string _init = "") const
Prints the current rational assignment.
```

**Parameters**

|              |                                      |
|--------------|--------------------------------------|
| <i>_out</i>  | The output stream to print on.       |
| <i>_init</i> | The beginning of each line to print. |

```
0.15.299.5.95 printReceivedFormula() void smtrat::Module::printReceivedFormula (
 const std::string & _initiation = "***") const [inherited]
Prints the vector of the received formula.
```

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.299.5.96 printTableau() template<class Settings >
void smtrat::LRAModule< Settings >::printTableau (
 std::ostream & _out = std::cout,
 const std::string _init = "") const
Prints the current tableau.
```

**Parameters**

|              |                                      |
|--------------|--------------------------------------|
| <i>_out</i>  | The output stream to print on.       |
| <i>_init</i> | The beginning of each line to print. |

---

```
0.15.299.5.97 printVariables() template<class Settings >
void smtrat::LRAgent< Settings >::printVariables (
 std::ostream & _out = std::cout,
 const std::string _init = "") const
```

Prints all lra variables and their assignments.

#### Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>_out</i>  | The output stream to print on.       |
| <i>_init</i> | The beginning of each line to print. |

```
0.15.299.5.98 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.299.5.99 processLearnedBounds() template<class Settings >
void smtrat::LRAgent< Settings >::processLearnedBounds ()
```

```
0.15.299.5.100 processResult() template<class Settings >
Answer smtrat::LRAgent< Settings >::processResult (
 Answer _result)
```

```
0.15.299.5.101 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline],
[inherited]
```

Notifies that the received formulas has been checked.

```
0.15.299.5.102 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs<→
InfeasibleSubset (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.299.5.103 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

**0.15.299.5.104 remove()** `void smtrat::Module::remove (`  
 `ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.299.5.105 removeCore()** `template<class Settings >`  
`void smtrat::LRAbstractModule< Settings >::removeCore (`  
 `ModuleInput::const_iterator _subformula ) [virtual]`

Removes everything related to the given sub-formula of the received formula.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.299.5.106 removeOrigin()** `std::pair<ModuleInput::iterator, bool> smtrat::Module::remove←`  
`Origin (`  
 `ModuleInput::iterator _formula,`  
 `const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.299.5.107 removeOrigins()** `std::pair<ModuleInput::iterator, bool> smtrat::Module::remove←`  
`Origins (`  
 `ModuleInput::iterator _formula,`  
 `const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],`  
`[inherited]`

**0.15.299.5.108 rPassedFormula()** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const`  
`[inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.299.5.109 rReceivedFormula()** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const`  
`[inline], [inherited]`

#### Returns

A reference to the formula passed to this module.

**0.15.299.5.110 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends ( ) [inline],`  
`[protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

---

**0.15.299.5.111 runBackends()** [2/2] [Answer](#) smtrat::Module::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.299.5.112 setId()** void smtrat::Module::setId (

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

#### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <code>id</code> | The id to set this module's id to. |
|-----------------|------------------------------------|

**0.15.299.5.113 setThreadPriority()** void smtrat::Module::setThreadPriority (

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.299.5.114 slackVariables()** template<class Settings >

```
const ExVariableMap& smtrat::LRAModule< Settings >::slackVariables () const [inline]
```

#### Returns

A constant reference to the slack variables.

**0.15.299.5.115 solverState()** [Answer](#) smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.299.5.116 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint )` [protected], [inherited]

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.299.5.117 `threadPriority()`** `thread_priority smtrat::Module::threadPriority () const` [inline], [inherited]

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.299.5.118 `updateAllModels()`** `void smtrat::Module::updateAllModels ()` [virtual], [inherited]

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.299.5.119 `updateLemmas()`** `void smtrat::Module::updateLemmas ()` [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.299.5.120 `updateModel()`** `template<class Settings >`

`void smtrat::LRAutoModule< Settings >::updateModel () const` [virtual]

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from [smtrat::Module](#).

**0.15.299.5.121 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends ()`

const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.299.6 Field Documentation

**0.15.299.6.1 `mAllBackends`** `std::vector<Module*> smtrat::Module::mAllBackends` [protected], [inherited]

The backends of this module which have been used.

**0.15.299.6.2 `mAllModels`** `std::vector<Model> smtrat::Module::mAllModels` [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.299.6.3 mBackendsFoundAnswer** `std::atomic_bool* smrat::Module::mBackendsFoundAnswer`

[protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.299.6.4 mConstraintsToInform** `carl::FastSet<FormulaT> smrat::Module::mConstraintsToInform`

[protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.299.6.5 mFinalCheck** `bool smrat::Module::mFinalCheck [protected], [inherited]`

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.299.6.6 mFirstPosInLastBranches** `std::size_t smrat::Module::mFirstPosInLastBranches = 0`

[static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.299.6.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.299.6.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.299.6.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer [protected], [inherited]`

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.299.6.10 mFullCheck** `bool smrat::Module::mFullCheck [protected], [inherited]`

false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.299.6.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smrat::Module::mInfeasibleSubsets`

[protected], [inherited]

Stores the infeasible subsets.

**0.15.299.6.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints`

[protected], [inherited]

Stores the position of the first constraint of which no backend has been informed about.

**0.15.299.6.13 mLastBranches** `std::vector< Branching > smrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`

Stores the last branches in a cycle buffer.

**0.15.299.6.14 mLemmas** std::vector<[Lemma](#)> smtrat::Module::mLemmas [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.299.6.15 mModel** [Model](#) smtrat::Module::mModel [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.299.6.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.299.6.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.299.6.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.299.6.19 mOldSplittingVariables** std::vector< [FormulaT](#) > smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.299.6.20 mpManager** [Manager](#)\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.299.6.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.299.6.22 mSolverState** std::atomic<[Answer](#)> smtrat::Module::mSolverState [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.299.6.23 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.299.6.24 mUsedBackends** std::vector<[Module](#)\*> smtrat::Module::mUsedBackends [protected], [inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.299.6.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.300 smtrat::LVEModule < Settings > Class Template Reference

```
#include <LVEModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
- `LVEModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)`
- `~LVEModule ()`
- `void updateModel () const`  
*Updates the current assignment into the model.*
- `Answer checkCore ()`  
*Checks the received formula for consistency.*
- `bool isPreprocessor () const`
- `bool appliedPreprocessing () const`
- `bool add (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `void remove (ModuleInput::const_iterator _subformula)`  
*Removes everything related to the given sub-formula of the received formula.*
- `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
*Checks the received formula for consistency.*
- `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- `virtual Answer runBackends ()`
- `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `bool inform (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void deinform (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- `virtual void init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `virtual void updateAllModels ()`  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`  
*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`  
*Sets the priority of this module to get a thread for running its check procedure.*
- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`

- const `Model & model () const`
- const `std::vector< Model > & allModels () const`
- const `std::vector< FormulaSetT > & infeasibleSubsets () const`
- const `std::vector< Module * > & usedBackends () const`
- const `carl::FastSet< FormulaT > & constraintsToInform () const`
- const `carl::FastSet< FormulaT > & informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const `std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- `carl::Variable objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `hasValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _insubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const FormulaT &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- virtual bool `addCore` (ModuleInput::const\_iterator formula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (ModuleInput::const\_iterator formula)  
*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- ModuleInput::iterator `passedFormulaBegin` ()
- ModuleInput::iterator `passedFormulaEnd` ()
- void `addOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const FormulaT & `getOrigins` (ModuleInput::const\_iterator \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const FormulaT &\_formula, FormulasT &\_origins) const
- void `getOrigins` (const FormulaT &\_formula, FormulaSetT &\_origins) const
- std::pair< ModuleInput::iterator, bool > `removeOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*ptr< std::vector< FormulaT >> &\_origins)*
- std::pair< ModuleInput::iterator, bool > `removeOrigins` (ModuleInput::iterator \_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)
- void `informBackends` (const FormulaT &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const FormulaT &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< ModuleInput::iterator, bool > `addReceivedSubformulaToPassedFormula` (ModuleInput::const\_iterator \_subformula)  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const FormulaT &\_origin) const
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula)  
*Adds the given formula to the passed formula with no origin.*

- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void [getBackendsModel](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
- void [getBackendsAllModels](#) () const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- virtual [ModuleInput::iterator](#) [eraseSubformulaFromPassedFormula](#) ([ModuleInput::iterator](#) \_subformula, bool \_ignoreOrigins=false)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void [clearPassedFormula](#) ()
 

*Merges the two vectors of sets into the first one.*
- std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
   
bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
   
bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
   
void [splitUnequalConstraint](#) (const [FormulaT](#) &)*

*Adds the following lemmas for the given constraint p!=0.*
- unsigned [checkModel](#) () const
 

*Adds a formula to the InformationRelevantFormula.*
- void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)
 

*Gets all InformationRelevantFormulas.*
- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector<FormulaSetT> mInfeasibleSubsets`

*Stores the infeasible subsets.*

- `Manager *const mpManager`

*A reference to the manager.*

- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*

- `std::vector<Model> mAllModels`

*Stores all satisfying assignments.*

- `bool mModelComputed`

*True, if the model has already been computed.*

- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*

- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*

- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*

- `std::vector<TheoryPropagation> mTheoryPropagations`

- `std::atomic<Answer> mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- `Conditionals mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- `std::vector<Module *> mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- `std::vector<Module *> mAllBackends`

*The backends of this module which have been used.*

- `std::vector<Lemma> mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*

- `ModuleInput::iterator mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- `carl::FastSet<FormulaT> mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet<FormulaT> mInformedConstraints`

*Stores the position of the first constraint of which no backend has been informed about.*

- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- `unsigned mSmallerMusesCheckCounter`

*Counter used for the generation of the smt2 files to check for smaller muses.*

- `std::vector<std::size_t> mVariableCounters`

*Maps variables to the number of their occurrences.*

### 0.15.300.1 Member Typedef Documentation

**0.15.300.1.1 SettingsType** template<typename Settings >  
 typedef `Settings smtrat::LVEModule< Settings >::SettingsType`

### 0.15.300.2 Member Enumeration Documentation

**0.15.300.2.1 LemmaType** enum `smtrat::Module::LemmaType` : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.300.3 Constructor & Destructor Documentation

**0.15.300.3.1 LVEModule()** template<typename Settings >  
`smtrat::LVEModule< Settings >::LVEModule` (  
 const `ModuleInput` \* `_formula`,  
`Conditionals` & `_conditionals`,  
`Manager` \* `_manager` = `nullptr` )

**0.15.300.3.2 ~LVEModule()** template<typename Settings >  
`smtrat::LVEModule< Settings >::~LVEModule` ( )

### 0.15.300.4 Member Function Documentation

**0.15.300.4.1 add()** bool `smtrat::PModule::add` (  
`ModuleInput::const_iterator` `_subformula`) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.300.4.2 addConstraintToInform()** void `smtrat::Module::addConstraintToInform` (  
`const FormulaT` & `_constraint`) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                    |                        |
|--------------------|------------------------|
| <i>_constraint</i> | The constraint to add. |
|--------------------|------------------------|

Reimplemented in [smrat::SATModule< Settings >](#).

**0.15.300.4.3 addCore()** `virtual bool smrat::Module::addCore ( ModuleInput::const_iterator formula ) [inline], [protected], [virtual], [inherited]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>formula</i> | The sub-formula to take additionally into account. |
|----------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.300.4.4 addInformationRelevantFormula()** `void smrat::Module::addInformationRelevantFormula (`

`const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.300.4.5 addLemma()** `void smrat::Module::addLemma (`

`const FormulaT & _lemma,`

`const LemmaType & _lt = LemmaType::NORMAL,`

`const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],`

`[inherited]`

Stores a lemma being a valid formula.

**Parameters**

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.300.4.6 addOrigin()** void smtrat::Module::addOrigin (   
     ModuleInput::iterator \_formula,  
     const FormulaT & \_origin ) [inline], [protected], [inherited]  
 Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

**0.15.300.4.7 addReceivedSubformulaToPassedFormula()** std::pair<ModuleInput::iterator,bool> smtrat::Module::addReceivedSubformulaToPassedFormula (   
     ModuleInput::const\_iterator \_subformula ) [inline], [protected], [inherited]

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

**0.15.300.4.8 addSubformulaToPassedFormula() [1/3]** std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (   
     const FormulaT & \_formula ) [inline], [protected], [inherited]

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.300.4.9 addSubformulaToPassedFormula() [2/3]** std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (   
     const FormulaT & \_formula,  
     const FormulaT & \_origin ) [inline], [protected], [inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                 |
| _origin  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.300.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>**

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.300.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]****Returns**

All satisfying assignments, if existent.

**0.15.300.4.12 anAnswerFound() bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.300.4.13 answerFound() const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.300.4.14 appliedPreprocessing() bool smrat::PModule::appliedPreprocessing () const [inline], [inherited]****Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.300.4.15 `backendsModel()`** const `Model` & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.300.4.16 `branchAt() [1/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.300.4.17 `branchAt() [2/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.300.4.18 `branchAt() [3/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.300.4.19 `branchAt() [4/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [protected], [inherited]

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                            |                                                                                                                                                                                                                |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>   | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>     | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>        | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>      | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code> | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |

**Parameters**

|                                           |                                                                                              |
|-------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise. |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                |

**0.15.300.4.20 `check()`** `Answer smrat::PModule::check (`

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.300.4.21 `checkCore()`** `template<typename Settings >`

```
Answer smrat::LVEModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.300.4.22 `checkInfeasibleSubsetForMinimality()`** `void smrat::Module::checkInfeasibleSubsetForMinimality (`

```
 std::vector< FormulaSet >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.300.4.23 `checkModel()`** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.300.4.24 `cleanModel()`** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.300.4.25 `clearLemmas()`** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.300.4.26 `clearModel()`** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.300.4.27 `clearModels()`** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.300.4.28 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.300.4.29 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.300.4.30 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.300.4.31 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

**0.15.300.4.32 constraintsToInform()** const carl::FastSet<[FormulaT](#)>& smrat::Module::constraintsToInform( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.300.4.33 currentlySatisfied()** virtual unsigned smrat::Module::currentlySatisfied( const [FormulaT](#) & ) const [inline], [virtual], [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.300.4.34 currentlySatisfiedByBackend()** unsigned smrat::Module::currentlySatisfiedByBackend( const [FormulaT](#) & \_formula ) const [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.300.4.35 deinform()** void smrat::Module::deinform( const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.300.4.36 deinformCore()** virtual void smrat::Module::deinformCore( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.300.4.37 determine\_smallest\_origin()** size\_t smrat::Module::determine\_smallest\_origin( const std::vector< [FormulaT](#) > & origins ) const [protected], [inherited]

## Parameters

|                       |                              |
|-----------------------|------------------------------|
| <code>_origins</code> | A vector of sets of origins. |
|-----------------------|------------------------------|

## Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.300.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::erase<SubformulaFromPassedFormula (`

```
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

## Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

## Returns

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.300.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceived<VariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.300.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::find<BestOrigin (`

```
 const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

## Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

## Returns

**0.15.300.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformula<ToPass ( ) const [inline], [inherited]`

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.300.4.42 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.300.4.43 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.300.4.44 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.300.4.45 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.300.4.46 getBackendsModel()** `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.300.4.47 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.300.4.48 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (`

`const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.300.4.49 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.300.4.50 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.300.4.51 `getOrigins() [1/3]`** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulaSetT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.300.4.52 `getOrigins() [2/3]`** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulasT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.300.4.53 `getOrigins() [3/3]`** `const FormulaT& smtrat::Module::getOrigins (`

```
ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.300.4.54 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.300.4.55 hasLemmas()** `bool smrat::Module::hasLemmas () [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.300.4.56 hasValidInfeasibleSubset()** `bool smrat::Module::hasValidInfeasibleSubset () const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.300.4.57 id()** `std::size_t smrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.300.4.58 infeasibleSubsets()** `const std::vector<FormulaSetT>& smrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.300.4.59 inform()** `bool smrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.300.4.60 informBackends()** `void smrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.300.4.61 informCore()** `virtual bool smrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

`false`, if it can be easily decided whether the given constraint is inconsistent; `true`, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

#### 0.15.300.4.62 informedConstraints()

```
const carl::FastSet<FormulaT>& smrat::Module::informed<→
Constraints () const [inline], [inherited]
```

#### Returns

The position of the first constraint of which no backend has been informed about.

#### 0.15.300.4.63 init()

```
void smrat::Module::init () [virtual], [inherited]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

#### 0.15.300.4.64 is\_minimizing()

```
bool smrat::Module::is_minimizing () const [inline], [inherited]
```

#### 0.15.300.4.65 isLemmaLevel()

```
bool smrat::Module::isLemmaLevel (
 LemmaLevel level) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

#### 0.15.300.4.66 isPreprocessor()

```
bool smrat::PModule::isPreprocessor () const [inline], [inherited]
```

#### Returns

`true`, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.300.4.67 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.300.4.68 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & [\\_vecSetA](#), const std::vector< [FormulaT](#) > & [\\_vecSetB](#) ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                          |                                  |
|--------------------------|----------------------------------|
| <a href="#">_vecSetA</a> | A vector of sets of constraints. |
| <a href="#">_vecSetB</a> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.300.4.69 model()** const [Model](#)& smtrat::Module::model ( ) const [inline], [inherited]

**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.300.4.70 modelsDisjoint()** bool smtrat::Module::modelsDisjoint ( const [Model](#) & [\\_modelA](#), const [Model](#) & [\\_modelB](#) ) [static], [protected], [inherited]

Checks whether there is no variable assigned by both models.

**Parameters**

|                         |                                |
|-------------------------|--------------------------------|
| <a href="#">_modelA</a> | The first model to check for.  |
| <a href="#">_modelB</a> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.300.4.71 moduleName()** template<typename Settings > std::string smtrat::LVEModule< [Settings](#) >::moduleName ( ) const [inline], [virtual]

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.300.4.72 `objective()`** `carl::Variable smrat::Module::objective ( ) const [inline], [inherited]`

**0.15.300.4.73 `originInReceivedFormula()`** `bool smrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.300.4.74 `passedFormulaBegin()`** `ModuleInput::iterator smrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.300.4.75 `passedFormulaEnd()`** `ModuleInput::iterator smrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.300.4.76 `pPassedFormula()`** `const ModuleInput* smrat::Module::pPassedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.300.4.77 `pReceivedFormula()`** `const ModuleInput* smrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.300.4.78 `print()`** `void smrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.300.4.79 `printAllModels()`** `void smrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.300.4.80 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the infeasible subsets.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.300.4.81 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]  
Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.300.4.82 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the vector of passed formula.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.300.4.83 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the vector of the received formula.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.300.4.84 probablyLooping()** bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & *\_branchingPolynomial*, const Rational & *\_branchingValue* ) const [protected], [inherited]  
Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

**Returns**

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.300.4.85 receivedFormulaChecked()** void smrat::Module::receivedFormulaChecked () [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.300.4.86 receivedFormulasAsInfeasibleSubset()** void smrat::Module::receivedFormulasAsInfeasibleSubset (

    ModuleInput::const\_iterator \_subformula ) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

**0.15.300.4.87 receivedVariable()** bool smrat::Module::receivedVariable (

    carl::Variable::Arg \_var ) const [inline], [inherited]

**0.15.300.4.88 remove()** void smrat::PModule::remove (

    ModuleInput::const\_iterator \_subformula ) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub formula of the received formula to remove. |
|-------------|----------------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.300.4.89 removeCore()** virtual void smrat::Module::removeCore (

    ModuleInput::const\_iterator formula ) [inline], [protected], [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|         |                                                    |
|---------|----------------------------------------------------|
| formula | The sub formula of the received formula to remove. |
|---------|----------------------------------------------------|

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.300.4.90 removeOrigin()** std::pair<ModuleInput::iterator, bool> smrat::Module::removeOrigin (

```
Origin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.300.4.91 removeOrigins()** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.300.4.92 rPassedFormula()** const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.300.4.93 rReceivedFormula()** const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.300.4.94 runBackends() [1/2]** virtual Answer smtrat::PModule::runBackends ( ) [inline], [virtual], [inherited]

Reimplemented from [smtrat::Module](#).

**0.15.300.4.95 runBackends() [2/2]** Answer smtrat::PModule::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.300.4.96 setId()** void smtrat::Module::setId (

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

## Parameters

|                         |                                    |
|-------------------------|------------------------------------|
| $\leftrightarrow$       | The id to set this module's id to. |
| $\underline{\text{id}}$ |                                    |

**0.15.300.4.97 `setThreadPriority()`** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority )` [inline], [inherited]

Sets the priority of this module to get a thread for running its check procedure.

## Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.300.4.98 `solverState()`** `Answer smtrat::Module::solverState ( ) const` [inline], [inherited]

## Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.300.4.99 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint )` [protected], [inherited]

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

## Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.300.4.100 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const` [inline], [inherited]

## Returns

The priority of this module to get a thread for running its check procedure.

**0.15.300.4.101 `updateAllModels()`** `void smtrat::Module::updateAllModels ( )` [virtual], [inherited]

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.300.4.102 `updateLemmas()`** `void smtrat::Module::updateLemmas ( )` [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.300.4.103 updateModel()** template<typename Settings>  
void smrat::LVEModule<Settings>::updateModel() const [virtual]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smrat::Module](#).

**0.15.300.4.104 usedBackends()** const std::vector<[Module](#)\*>& smrat::Module::usedBackends() const [inline], [inherited]

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.300.5 Field Documentation

**0.15.300.5.1 mAllBackends** std::vector<[Module](#)\*> smrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

**0.15.300.5.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.300.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer [protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.300.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.300.5.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.300.5.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.300.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.300.5.8 mFirstUncheckedReceivedSubformula** [ModuleInput](#)::const\_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.300.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.300.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.300.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.300.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.300.5.13 mLastBranches** `std::vector<Branching>` `smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.300.5.14 mLemmas** `std::vector<Lemma>` `smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.300.5.15 mModel** `Model` `smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.300.5.16 mModelComputed** `bool` `smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.300.5.17 mNumOfBranchVarsToStore** `std::size_t` `smtrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.300.5.18 mObjectiveVariable** `carl::Variable` `smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.300.5.19 mOldSplittingVariables** `std::vector<FormulaT>` `smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.300.5.20 mpManager** `Manager*` `const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.300.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter`  
[mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.300.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected],  
[inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.300.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheory←Propagations` [protected], [inherited]

**0.15.300.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends` [protected],  
[inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.300.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters`  
[protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.301 smtrat::Manager Class Reference

Base class for solvers.

```
#include <Manager.h>
```

### Public Member Functions

- **Manager ()**  
*Constructs a manager.*
- **~Manager ()**  
*Destructs a manager.*
- **bool inform (const FormulaT &\_constraint)**  
*Informs the solver about a constraint.*
- **void deinform (const FormulaT &\_constraint)**  
*The inverse of informing about a constraint.*
- **bool add (const FormulaT &\_subformula, bool \_containsUnknownConstraints=true)**  
*Adds the given formula to the conjunction of formulas, which will be considered for the next satisfiability check.*
- **Answer check (bool \_full=true)**  
*Checks the so far added formulas for satisfiability.*
- **void push ()**  
*Pushes a backtrack point to the stack of backtrack points.*
- **bool pop ()**  
*Pops a backtrack point from the stack of backtrack points.*
- **void pop (size\_t \_levels)**
- **void clear ()**
- **void setObjectiveVariable (carl::Variable var)**
- **const carl::Variable & objectiveVariable () const**
- **void reset ()**
- **const std::vector<FormulaSetT> & infeasibleSubsets () const**
- **std::list<std::vector<carl::Variable>> getModelEqualities () const**

- `const Model & model () const`  
*Determines variables assigned by the currently found satisfying assignment to an equal value in their domain.*
- `const std::vector< Model > & allModels () const`
- `std::vector< FormulaT > lemmas ()`  
*Returns the lemmas/tautologies which were made during the last solving provoked by `check()`.*
- `const ModuleInput & formula () const`
- `ModuleInput::iterator formulaBegin ()`
- `ModuleInput::iterator formulaEnd ()`
- `void printAssignment () const`  
*Prints the currently found assignment of variables occurring in the so far added formulas to values of their domains, if the conjunction of these formulas is satisfiable.*
- `void printAllAssignments (std::ostream & _out=std::cout)`  
*Prints all assignments of variables occurring in the so far added formulas to values of their domains, if the conjunction of these formulas is satisfiable.*
- `void printInfeasibleSubset (std::ostream & _out=std::cout) const`  
*Prints the first found infeasible subset of the set of received formulas.*
- `void printBackTrackStack (std::ostream & _out=std::cout) const`  
*Prints the stack of backtrack points.*
- `void printStrategyGraph (std::ostream & _out=std::cout) const`  
*Prints the strategy of the solver maintained by this manager.*
- `const std::vector< Module * > & getAllGeneratedModules () const`
- `std::ostream & rDebugOutputChannel ()`
- `auto logic () const`
- `auto & rLogic ()`
- `void addInformationRelevantFormula (const FormulaT & formula)`  
*Adds formula to `InformationRelevantFormula`.*
- `void clearInformationRelevantFormula ()`  
*Deletes all `InformationRelevantFormula`.*
- `void setLemmaLevel (LemmaLevel level)`  
*Sets the current level for lemma generation.*
- `bool isLemmaLevel (LemmaLevel level)`  
*Checks if current lemma level is greater or equal to given level.*
- `std::pair< bool, FormulaT > getInputSimplified ()`
- `ModuleInput::iterator remove (ModuleInput::iterator _subformula)`  
*Removes the formula at the given position in the conjunction of formulas, which will be considered for the next satisfiability check.*
- `ModuleInput::iterator remove (const FormulaT & _subformula)`  
*Temporarily added: (TODO: Discuss with Gereon) Removes the given formula in the conjunction of formulas, which will be considered for the next satisfiability check.*

## Protected Member Functions

- `void setStrategy (const std::initializer_list< BackendLink > & backends)`
- `void setStrategy (const BackendLink & backend)`
- `template<typename Module> BackendLink addBackend (const std::initializer_list< BackendLink > & backends={})`
- `template<typename Module> BackendLink addBackend (const BackendLink & backend)`
- `BackendLink & addEdge (std::size_t from, std::size_t to)`
- `std::vector< Module * > getAllBackends (Module *_module) const`  
*Gets all backends so far instantiated according the strategy and all previous enquiries of the given module.*
- `std::vector< Module * > getBackends (Module *_requiredBy, std::atomic_bool *_foundAnswer)`  
*Get the backends to call for the given problem instance required by the given module.*
- `const std::set< FormulaT > & getInformationRelevantFormulas ()`  
*Gets all `InformationRelevantFormulas`.*

**Friends**

- class [Module](#)

**0.15.301.1 Detailed Description**

Base class for solvers.

This is the interface to the user.

**0.15.301.2 Constructor & Destructor Documentation****0.15.301.2.1 Manager()** `smtrat::Manager::Manager ()`

Constructs a manager.

**0.15.301.2.2 ~Manager()** `smtrat::Manager::~Manager ()`

Destructs a manager.

**0.15.301.3 Member Function Documentation****0.15.301.3.1 add()** `bool smtrat::Manager::add (`  
`const FormulaT & _subformula,`  
`bool _containsUnknownConstraints = true )`

Adds the given formula to the conjunction of formulas, which will be considered for the next satisfiability check.

**Parameters**

|                                          |                                                                                                 |
|------------------------------------------|-------------------------------------------------------------------------------------------------|
| <code>_subformula</code>                 | The formula to add.                                                                             |
| <code>_containsUnknownConstraints</code> | true, if the formula to add contains constraints, about which this solver was not yet informed. |

**Returns**

false, if it is easy to decide whether adding this formula creates a conflict; true, otherwise.

**0.15.301.3.2 addBackend() [1/2]** `template<typename Module >`  
`BackendLink smtrat::Manager::addBackend (`  
`const BackendLink & backend ) [inline], [protected]`**0.15.301.3.3 addBackend() [2/2]** `template<typename Module >`  
`BackendLink smtrat::Manager::addBackend (`  
`const std::initializer_list< BackendLink > & backends = {} ) [inline], [protected]`**0.15.301.3.4 addEdge()** `BackendLink& smtrat::Manager::addEdge (`  
`std::size_t from,`  
`std::size_t to ) [inline], [protected]`

**0.15.301.3.5 addInformationRelevantFormula()** void smtrat::Manager::addInformationRelevantFormula ( const FormulaT & formula ) [inline]  
 Adds formula to InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.301.3.6 allModels()** const std::vector<Model>& smtrat::Manager::allModels ( ) const [inline]

**Returns**

A list of all assignments, such that they satisfy the conjunction of the so far added formulas.

Note, that an assignment is only provided if the conjunction of so far added formulas is satisfiable. Furthermore, when solving non-linear real arithmetic formulas the assignment could contain other variables or freshly introduced variables.

**0.15.301.3.7 check()** Answer smtrat::Manager::check ( bool \_full = true )

Checks the so far added formulas for satisfiability.

**Returns**

True, if the conjunction of the so far added formulas is satisfiable; False, if it not satisfiable; Unknown, if this solver cannot decide whether it is satisfiable or not.

**0.15.301.3.8 clear()** void smtrat::Manager::clear ( ) [inline]

**0.15.301.3.9 clearInformationRelevantFormula()** void smtrat::Manager::clearInformationRelevantFormula ( ) [inline]  
 Deletes all InformationRelevantFormula.

**0.15.301.3.10 deinform()** void smtrat::Manager::deinform ( const FormulaT & \_constraint )

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                    |                                                         |
|--------------------|---------------------------------------------------------|
| <i>_constraint</i> | The constraint to remove from internal data structures. |
|--------------------|---------------------------------------------------------|

**0.15.301.3.11 formula()** const ModuleInput& smtrat::Manager::formula ( ) const [inline]

**Returns**

The conjunction of so far added formulas.

**0.15.301.3.12 formulaBegin()** `ModuleInput::iterator smrat::Manager::formulaBegin ( ) [inline]`

**0.15.301.3.13 formulaEnd()** `ModuleInput::iterator smrat::Manager::formulaEnd ( ) [inline]`

**0.15.301.3.14 getAllBackends()** `std::vector<Module*> smrat::Manager::getAllBackends ( Module * _module ) const [inline], [protected]`

Gets all backends so far instantiated according the strategy and all previous enquiries of the given module.

**Parameters**

|                      |                                                                                                                      |
|----------------------|----------------------------------------------------------------------------------------------------------------------|
| <code>_module</code> | The module to get all backends so far instantiated according the strategy and all previous enquiries of this module. |
|----------------------|----------------------------------------------------------------------------------------------------------------------|

**Returns**

All backends so far instantiated according the strategy and all previous enquiries of the given module.

**0.15.301.3.15 getAllGeneratedModules()** `const std::vector<Module*>& smrat::Manager::getAllGeneratedModules ( ) const [inline]`

**Returns**

All instantiated modules of the solver belonging to this manager.

**0.15.301.3.16 getBackends()** `std::vector<Module * > smrat::Manager::getBackends ( Module * _requiredBy, std::atomic_bool * _foundAnswer ) [protected]`

Get the backends to call for the given problem instance required by the given module.

**Parameters**

|                           |                                  |
|---------------------------|----------------------------------|
| <code>_requiredBy</code>  | The module asking for a backend. |
| <code>_foundAnswer</code> | A conditional                    |

**Returns**

A vector of modules, which the module defined by `_requiredBy` calls in parallel to achieve an answer to the given instance.

**0.15.301.3.17 getInformationRelevantFormulas()** `const std::set<FormulaT>& smrat::Manager::getInformationRelevantFormulas ( ) [inline], [protected]`

Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.301.3.18 getInputSimplified()** `std::pair< bool, FormulaT > smrat::Manager::getInputSimplified ( )`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the input formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this solver's input formula, if the Boolean is true.

**0.15.301.3.19 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Manager::getModelEqualities( ) const`  
 Determines variables assigned by the currently found satisfying assignment to an equal value in their domain.

**Returns**

A list of vectors of variables, stating that the variables in one vector are assigned to equal values.

**0.15.301.3.20 `infeasibleSubsets()`** `const std::vector< FormulaSetT > & smtrat::Manager::infeasibleSubsets( ) const`

**Returns**

All infeasible subsets of the set so far added formulas.

Note, that the conjunction of the so far added formulas must be inconsistent to receive an infeasible subset.

**0.15.301.3.21 `inform()`** `bool smtrat::Manager::inform( const FormulaT & _constraint )`

Informs the solver about a constraint.

Optimally, it should be informed about all constraints, which it will receive eventually, before any of them is added as part of a formula with the interface add(..).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it is easy to decide (for any module used of this solver), whether the constraint itself is inconsistent; true, otherwise.

**0.15.301.3.22 `isLemmaLevel()`** `bool smtrat::Manager::isLemmaLevel( LemmaLevel level ) [inline]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.301.3.23 `lemmas()`** `std::vector< FormulaT > smtrat::Manager::lemmas( )`  
 Returns the lemmas/tautologies which were made during the last solving provoked by [check\(\)](#).  
 These lemmas can be used in the same manner as infeasible subsets are used.

**Returns**

The lemmas/tautologies made during solving.

**0.15.301.3.24 logic()** auto smtrat::Manager::logic ( ) const [inline]

Returns

A constant reference to the logic this solver considers.

**0.15.301.3.25 model()** const Model & smtrat::Manager::model ( ) const

Returns

An assignment of the variables, which occur in the so far added formulas, to values of their domains, such that it satisfies the conjunction of these formulas.

Note, that an assignment is only provided if the conjunction of so far added formulas is satisfiable. Furthermore, when solving non-linear real arithmetic formulas the assignment could contain other variables or freshly introduced variables.

**0.15.301.3.26 objectiveVariable()** const carl::Variable& smtrat::Manager::objectiveVariable ( ) const [inline]

**0.15.301.3.27 pop() [1/2]** bool smtrat::Manager::pop ( void )

Pops a backtrack point from the stack of backtrack points.

This provokes, that all formulas which have been added after that backtrack point are removed.

Note, that this interface has not necessarily be used to apply a solver constructed with SMT-RAT, but is often required by state-of-the-art SMT solvers when embedding a theory solver constructed with SMT-RAT into them.

**0.15.301.3.28 pop() [2/2]** void smtrat::Manager::pop ( size\_t \_levels )

**0.15.301.3.29 printAllAssignments()** void smtrat::Manager::printAllAssignments ( std::ostream & \_out = std::cout ) [inline]

Prints all assignments of variables occurring in the so far added formulas to values of their domains, if the conjunction of these formulas is satisfiable.

Parameters

|             |                         |
|-------------|-------------------------|
| <i>_out</i> | The stream to print on. |
|-------------|-------------------------|

**0.15.301.3.30 printAssignment()** void smtrat::Manager::printAssignment ( ) const

Prints the currently found assignment of variables occurring in the so far added formulas to values of their domains, if the conjunction of these formulas is satisfiable.

**0.15.301.3.31 printBackTrackStack()** void smtrat::Manager::printBackTrackStack ( std::ostream & \_out = std::cout ) const

Prints the stack of backtrack points.

Parameters

|             |                         |
|-------------|-------------------------|
| <i>_out</i> | The stream to print on. |
|-------------|-------------------------|

```
0.15.301.3.32 printInfeasibleSubset() void smtrat::Manager::printInfeasibleSubset (
 std::ostream & _out = std::cout) const
```

Prints the first found infeasible subset of the set of received formulas.

#### Parameters

|                   |                         |
|-------------------|-------------------------|
| <code>_out</code> | The stream to print on. |
|-------------------|-------------------------|

```
0.15.301.3.33 printStrategyGraph() void smtrat::Manager::printStrategyGraph (
 std::ostream & _out = std::cout) const [inline]
```

Prints the strategy of the solver maintained by this manager.

#### Parameters

|                   |                         |
|-------------------|-------------------------|
| <code>_out</code> | The stream to print on. |
|-------------------|-------------------------|

```
0.15.301.3.34 push() void smtrat::Manager::push (
 void) [inline]
```

Pushes a backtrack point to the stack of backtrack points.

Note, that this interface has not necessarily be used to apply a solver constructed with SMT-RAT, but is often required by state-of-the-art SMT solvers when embedding a theory solver constructed with SMT-RAT into them.

```
0.15.301.3.35 rDebugOutputChannel() std::ostream& smtrat::Manager::rDebugOutputChannel () [inline]
```

#### Returns

The stream to print the debug information on.

```
0.15.301.3.36 remove() [1/2] ModuleInput::iterator smtrat::Manager::remove (
 const FormulaT & _subformula) [inline]
```

Temporarily added: (TODO: Discuss with Gereon) Removes the given formula in the conjunction of formulas, which will be considered for the next satisfiability check.

#### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_subformula</code> | The formula to remove. |
|--------------------------|------------------------|

#### Returns

An iterator to the formula after the position of the just removed formula. If the removed formula has been the last element, the end of the conjunction of formulas, which will be considered for the next satisfiability check is returned.

```
0.15.301.3.37 remove() [2/2] ModuleInput::iterator smtrat::Manager::remove (
 ModuleInput::iterator _subformula)
```

Removes the formula at the given position in the conjunction of formulas, which will be considered for the next satisfiability check.

**Parameters**

|                          |                                        |
|--------------------------|----------------------------------------|
| <code>_subformula</code> | The position of the formula to remove. |
|--------------------------|----------------------------------------|

**Returns**

An iterator to the formula after the position of the just removed formula. If the removed formula has been the last element, the end of the conjunction of formulas, which will be considered for the next satisfiability check is returned.

**0.15.301.3.38 `reset()`** `void smtrat::Manager::reset ( )`**0.15.301.3.39 `rLogic()`** `auto& smtrat::Manager::rLogic ( ) [inline]`**Returns**

A reference to the logic this solver considers.

**0.15.301.3.40 `setLemmaLevel()`** `void smtrat::Manager::setLemmaLevel ( LemmaLevel level ) [inline]`

Sets the current level for lemma generation.

**Parameters**

|                    |       |
|--------------------|-------|
| <code>level</code> | Level |
|--------------------|-------|

**0.15.301.3.41 `setObjectiveVariable()`** `void smtrat::Manager::setObjectiveVariable ( carl::Variable var ) [inline]`**0.15.301.3.42 `setStrategy()` [1/2]** `void smtrat::Manager::setStrategy ( const BackendLink & backend ) [inline], [protected]`**0.15.301.3.43 `setStrategy()` [2/2]** `void smtrat::Manager::setStrategy ( const std::initializer_list< BackendLink > & backends ) [inline], [protected]`**0.15.301.4 Friends And Related Function Documentation****0.15.301.4.1 `Module`** `friend class Module [friend]`**0.15.302 Minisat::Map< K, D, H, E > Class Template Reference**

```
#include <Map.h>
```

**Data Structures**

- struct `Pair`

## Public Member Functions

- `Map ()`
- `Map (const H &h, const E &e)`
- `~Map ()`
- `const D & operator[] (const K &k) const`
- `D & operator[] (const K &k)`
- `void insert (const K &k, const D &d)`
- `bool peek (const K &k, D &d) const`
- `bool has (const K &k) const`
- `void remove (const K &k)`
- `void clear ()`
- `int elems () const`
- `int bucket_count () const`
- `void moveTo (Map &other)`
- `const vec< Pair > & bucket (int i) const`

### 0.15.302.1 Constructor & Destructor Documentation

**0.15.302.1.1 `Map()` [1/2]** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
`Minisat::Map< K, D, H, E >::Map ( )` [inline]

**0.15.302.1.2 `Map()` [2/2]** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
`Minisat::Map< K, D, H, E >::Map (`  
`const H & h,`  
`const E & e )` [inline]

**0.15.302.1.3 `~Map()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
`Minisat::Map< K, D, H, E >::~Map ( )` [inline]

### 0.15.302.2 Member Function Documentation

**0.15.302.2.1 `bucket()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
`const vec<Pair>& Minisat::Map< K, D, H, E >::bucket (`  
`int i ) const` [inline]

**0.15.302.2.2 `bucket_count()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
`int Minisat::Map< K, D, H, E >::bucket_count ( ) const` [inline]

**0.15.302.2.3 `clear()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
`void Minisat::Map< K, D, H, E >::clear ( )` [inline]

**0.15.302.2.4 `elems()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
`int Minisat::Map< K, D, H, E >::elems ( ) const` [inline]

---

**0.15.302.2.5 `has()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
bool Minisat::Map< K, D, H, E >::has (  
    const K & k ) const [inline]

**0.15.302.2.6 `insert()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
void Minisat::Map< K, D, H, E >::insert (  
    const K & k,  
    const D & d ) [inline]

**0.15.302.2.7 `moveTo()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
void Minisat::Map< K, D, H, E >::moveTo (  
    Map< K, D, H, E > & other ) [inline]

**0.15.302.2.8 `operator[]()` [1/2]** template<class K , class D , class H = Hash<K>, class E = Equal<↔  
K>>  
D& Minisat::Map< K, D, H, E >::operator[] (   
    const K & k ) [inline]

**0.15.302.2.9 `operator[]()` [2/2]** template<class K , class D , class H = Hash<K>, class E = Equal<↔  
K>>  
const D& Minisat::Map< K, D, H, E >::operator[] (   
    const K & k ) const [inline]

**0.15.302.2.10 `peek()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
bool Minisat::Map< K, D, H, E >::peek (  
    const K & k,  
    D & d ) const [inline]

**0.15.302.2.11 `remove()`** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
void Minisat::Map< K, D, H, E >::remove (   
    const K & k ) [inline]

## 0.15.303 **benchmax::MathSAT Class Reference**

Tool wrapper for [MathSAT](#) for SMT-LIB.

```
#include <MathSAT.h>
```

### Public Member Functions

- **MathSAT** (const fs::path &[binary](#), const std::string &[arguments](#))  
*New MathSAT tool.*
- virtual bool [canHandle](#) (const fs::path &[path](#)) const override  
*Only handle .smt2 files.*
- virtual void [additionalResults](#) (const fs::path &, [BenchmarkResult](#) &[result](#)) const override  
*Computes the answer string from the exit code and parses statistics output.*
- std::string [name](#) () const  
*Common name of this tool.*
- fs::path [binary](#) () const  
*Full path to the binary.*

- const std::map< std::string, std::string > & [attributes](#) () const  
*A set of attributes, for example compilation options.*
- std::vector< std::string > [resolveDependencies](#) () const  
*Get dependencies of binary required to run it (via ldd)*
- std::size\_t [attributeHash](#) () const  
*Hash of the attributes.*
- virtual std::string [getCommandline](#) (const std::string &file) const  
*Compose commandline for this tool and the given input file.*
- virtual std::string [getCommandline](#) (const std::string &file, const std::string &localBinary) const  
*Compose commandline for this tool with another binary name and the given input file.*
- virtual std::optional< std::string > [parseCommandline](#) (const std::string &cmdline) const  
*Compose commandline for this tool and the given input file.*

## Protected Attributes

- std::string [mName](#)  
*(Non-unique) identifier for the tool.*
- fs::path [mBinary](#)  
*Path to the binary.*
- std::string [mArguments](#)  
*Command line arguments that should be passed to the binary.*
- std::map< std::string, std::string > [mAttributes](#)  
*Attributes of the tool obtained by introspection of the binary.*

### 0.15.303.1 Detailed Description

Tool wrapper for [MathSAT](#) for SMT-LIB.

### 0.15.303.2 Constructor & Destructor Documentation

**0.15.303.2.1 [MathSAT\(\)](#)** benchmax::MathSAT::MathSAT (

```
 const fs::path & binary,
 const std::string & arguments) [inline]
```

New [MathSAT](#) tool.

### 0.15.303.3 Member Function Documentation

**0.15.303.3.1 [additionalResults\(\)](#)** virtual void benchmax::MathSAT::additionalResults (

```
 const fs::path & ,
 BenchmarkResult & result) const [inline], [override], [virtual]
```

Computes the answer string from the exit code and parses statistics output.

Reimplemented from [benchmax::Tool](#).

**0.15.303.3.2 [attributeHash\(\)](#)** std::size\_t benchmax::Tool::attributeHash () const [inline], [inherited]

Hash of the attributes.

**0.15.303.3.3 attributes()** const std::map<std::string, std::string>& benchmax::Tool::attributes () const [inline], [inherited]  
A set of attributes, for example compilation options.

**0.15.303.3.4 binary()** fs::path benchmax::Tool::binary () const [inline], [inherited]  
Full path to the binary.

**0.15.303.3.5 canHandle()** virtual bool benchmax::MathSAT::canHandle (const fs::path & path) const [inline], [override], [virtual]  
Only handle .smt2 files.  
Reimplemented from [benchmax::Tool](#).

**0.15.303.3.6 getCommandLine() [1/2]** virtual std::string benchmax::Tool::getCommandLine (const std::string & file) const [inline], [virtual], [inherited]  
Compose commandline for this tool and the given input file.

**0.15.303.3.7 getCommandLine() [2/2]** virtual std::string benchmax::Tool::getCommandLine (const std::string & file, const std::string & localBinary) const [inline], [virtual], [inherited]  
Compose commandline for this tool with another binary name and the given input file.

**0.15.303.3.8 name()** std::string benchmax::Tool::name () const [inline], [inherited]  
Common name of this tool.

**0.15.303.3.9 parseCommandLine()** virtual std::optional<std::string> benchmax::Tool::parseCommandline (const std::string & cmdline) const [inline], [virtual], [inherited]  
Compose commandline for this tool and the given input file.

**0.15.303.3.10 resolveDependencies()** std::vector<std::string> benchmax::Tool::resolveDependencies () const [inline], [inherited]  
Get dependencies of binary required to run it (via ldd)

## 0.15.303.4 Field Documentation

**0.15.303.4.1 mArguments** std::string benchmax::Tool::mArguments [protected], [inherited]  
Command line arguments that should be passed to the binary.

**0.15.303.4.2 mAttributes** std::map<std::string, std::string> benchmax::Tool::mAttributes [protected], [inherited]  
Attributes of the tool obtained by introspection of the binary.

**0.15.303.4.3 mBinary** fs::path benchmax::Tool::mBinary [protected], [inherited]  
Path to the binary.

**0.15.303.4.4 mName** std::string benchmax::Tool::mName [protected], [inherited]  
 (Non-unique) identifier for the tool.

## 0.15.304 smtrat::mcsat::fm::MaxSizeComparator Struct Reference

This heuristic chooses the explanation excluding the largest interval.

```
#include <ConflictGenerator.h>
```

### Public Member Functions

- bool `operator()` (const `Bound` &b1, const `Bound` &b2) const

### Data Fields

- bool `symmetric` = true

#### 0.15.304.1 Detailed Description

This heuristic chooses the explanation excluding the largest interval.

#### 0.15.304.2 Member Function Documentation

**0.15.304.2.1 operator()** bool smtrat::mcsat::fm::MaxSizeComparator::operator() (const `Bound` & b1, const `Bound` & b2) const [inline]

#### 0.15.304.3 Field Documentation

**0.15.304.3.1 symmetric** bool smtrat::mcsat::fm::MaxSizeComparator::symmetric = true

## 0.15.305 smtrat::MaxSMT< Solver, Strategy > Class Template Reference

```
#include <MaxSMT.h>
```

### Public Member Functions

- `MaxSMT` (Solver &s)
- void `add_soft_formula` (const `FormulaT` &formula, `Rational` weight, const std::string &id)
- void `remove_soft_formula` (const `FormulaT` &formula)
- void `reset` ()
- bool `active` () const
- std::tuple< `Answer`, `Model`, `ObjectiveValues` > `compute` ()

#### 0.15.305.1 Constructor & Destructor Documentation

**0.15.305.1.1 MaxSMT()** template<typename Solver, MaxSMTStrategy Strategy>  
`smtrat::MaxSMT< Solver, Strategy >::MaxSMT` (Solver & s) [inline]

#### 0.15.305.2 Member Function Documentation

**0.15.305.2.1 `active()`** template<typename Solver , MaxSMTStrategy Strategy>  
bool smtrat::MaxSMT< Solver, Strategy >::active ( ) const [inline]

**0.15.305.2.2 `add_soft_formula()`** template<typename Solver , MaxSMTStrategy Strategy>  
void smtrat::MaxSMT< Solver, Strategy >::add\_soft\_formula (   
    const FormulaT & formula,  
    Rational weight,  
    const std::string & id ) [inline]

**0.15.305.2.3 `compute()`** template<typename Solver , MaxSMTStrategy Strategy>  
std::tuple<Answer,Model,ObjectiveValues> smtrat::MaxSMT< Solver, Strategy >::compute ( )  
[inline]

**0.15.305.2.4 `remove_soft_formula()`** template<typename Solver , MaxSMTStrategy Strategy>  
void smtrat::MaxSMT< Solver, Strategy >::remove\_soft\_formula (   
    const FormulaT & formula ) [inline]

**0.15.305.2.5 `reset()`** template<typename Solver , MaxSMTStrategy Strategy>  
void smtrat::MaxSMT< Solver, Strategy >::reset ( ) [inline]

## 0.15.306 smtrat::maxsmt::MaxSMTBackend< Solver, Strategy > Class Template Reference

```
#include <MaxSMT.h>
```

### 0.15.307 smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU\_MALIK\_INCREMENTAL > Class Template Reference

```
#include <MaxSMT_FuMalikIncremental.h>
```

#### Public Member Functions

- `MaxSMTBackend (Solver &solver, const std::vector< FormulaT > &softClauses)`
- `Answer run ()`

#### 0.15.307.1 Constructor & Destructor Documentation

**0.15.307.1.1 `MaxSMTBackend()`** template<typename Solver >  
smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU\_MALIK\_INCREMENTAL >::MaxSMTBackend (   
    Solver & solver,  
    const std::vector< FormulaT > & softClauses ) [inline]

#### 0.15.307.2 Member Function Documentation

**0.15.307.2.1 `run()`** template<typename Solver >  
Answer smtrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::FU\_MALIK\_INCREMENTAL >::run ( )  
[inline]

### 0.15.308 smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR\_SEARCH > Class Template Reference

```
#include <MaxSMT_LinearSearch.h>
```

#### Public Member Functions

- `MaxSMTBackend (Solver &solver, const std::vector< FormulaT > &softClauses)`
- `Answer run ()`

#### 0.15.308.1 Constructor & Destructor Documentation

```
0.15.308.1.1 MaxSMTBackend() template<typename Solver >
smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >::MaxSMTBackend (
 Solver & solver,
 const std::vector< FormulaT > & softClauses) [inline]
```

#### 0.15.308.2 Member Function Documentation

```
0.15.308.2.1 run() template<typename Solver >
Answer smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::LINEAR_SEARCH >::run () [inline]
```

### 0.15.309 smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 > Class Template Reference

```
#include <MaxSMT_MSU3.h>
```

#### Public Member Functions

- `MaxSMTBackend (Solver &solver, const std::vector< FormulaT > &softClauses)`
- `Answer run ()`

#### 0.15.309.1 Constructor & Destructor Documentation

```
0.15.309.1.1 MaxSMTBackend() template<typename Solver >
smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >::MaxSMTBackend (
 Solver & solver,
 const std::vector< FormulaT > & softClauses) [inline]
```

#### 0.15.309.2 Member Function Documentation

```
0.15.309.2.1 run() template<typename Solver >
Answer smrat::maxsmt::MaxSMTBackend< Solver, MaxSMTStrategy::MSU3 >::run () [inline]
```

### 0.15.310 smrat::MCBModule< Settings > Class Template Reference

```
#include <MCBModule.h>
```

## Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

## Public Member Functions

- `MCBModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)`
- `~MCBModule ()`
- `Answer checkCore ()`  
*Checks the received formula for consistency.*
- `void updateModel () const`  
*Updates the model, if the solver has detected the consistency of the received formula, beforehand.*
- `bool isPreprocessor () const`
- `bool appliedPreprocessing () const`
- `bool add (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `void remove (ModuleInput::const_iterator _subformula)`  
*Removes everything related to the given sub-formula of the received formula.*
- `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
*Checks the received formula for consistency.*
- `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- `virtual Answer runBackends ()`
- `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `bool inform (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- `void deinform (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- `virtual void init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `virtual void updateAllModels ()`  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`  
*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`  
*Sets the priority of this module to get a thread for running its check procedure.*
- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`
- `const std::vector< Model > & allModels () const`
- `const std::vector< FormulaSetT > & infeasibleSubsets () const`
- `const std::vector< Module * > & usedBackends () const`

- const carl::FastSet< `FormulaT` > & `constraintsToInform` () const
- const carl::FastSet< `FormulaT` > & `informedConstraints` () const
- void `addLemma` (const `FormulaT` &\_lemma, const `LemmaType` &\_lt=`LemmaType::NORMAL`, const `FormulaT` &\_preferredFormula=`FormulaT(carl::FormulaType::TRUE)`)
 

*Stores a lemma being a valid formula.*
- bool `hasLemmas` ()
 

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas` ()
 

*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas` () const
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula` () const
- `ModuleInput::const_iterator firstSubformulaToPass` () const
- void `receivedFormulaChecked` ()
 

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals` & `answerFound` () const
- virtual std::string `moduleName` () const
- carl::Variable `objective` () const
- bool `is_minimizing` () const
- void `excludeNotReceivedVariablesFromModel` () const
 

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas` ()
 

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations` ()
- void `collectOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
 

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- bool `isValidInfeasibleSubset` () const
- void `checkInfSubsetForMinimality` (std::vector< `FormulaSetT` >::const\_iterator \_infsSubset, const std::string &\_filename="smaller\_muses", unsigned \_maxSizeDifference=1) const
 

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print` (const std::string &\_initiation="\*\*\*") const
 

*Prints everything relevant of the solver.*
- void `printReceivedFormula` (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of the received formula.*
- void `printPassedFormula` (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const
 

*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const
 

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)
 

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const FormulaT &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- virtual bool `addCore` (ModuleInput::const\_iterator formula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (ModuleInput::const\_iterator formula)  
*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- ModuleInput::iterator `passedFormulaBegin` ()
- ModuleInput::iterator `passedFormulaEnd` ()
- void `addOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const FormulaT & `getOrigins` (ModuleInput::const\_iterator \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const FormulaT &\_formula, FormulasT &\_origins) const
- void `getOrigins` (const FormulaT &\_formula, FormulaSetT &\_origins) const
- std::pair< ModuleInput::iterator, bool > `removeOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*ptr< std::vector< FormulaT >> &\_origins)*
- std::pair< ModuleInput::iterator, bool > `removeOrigins` (ModuleInput::iterator \_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)
- void `informBackends` (const FormulaT &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const FormulaT &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< ModuleInput::iterator, bool > `addReceivedSubformulaToPassedFormula` (ModuleInput::const\_iterator \_subformula)  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const FormulaT &\_origin) const
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula)  
*Adds the given formula to the passed formula with no origin.*

- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void [getBackendsModel](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
- virtual [ModuleInput::iterator](#) [eraseSubformulaFromPassedFormula](#) ([ModuleInput::iterator](#) \_subformula, bool \_ignoreOrigins=false)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void [clearPassedFormula](#) ()
 

*Merges the two vectors of sets into the first one.*
- size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool [probablyLooping](#) (const typename [Poly::PolyType](#) &\_branchingPolynomial, const [Rational](#) &\_branchingValue) const
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- void [splitUnequalConstraint](#) (const [FormulaT](#) &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned [checkModel](#) () const
 

*Gets all InformationRelevantFormulas.*
- void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()
 

*Gets all InformationRelevantFormulas.*
- bool [isLemmaLevel](#) ([LemmaLevel](#) level)
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector<FormulaSetT> mInfeasibleSubsets`

*Stores the infeasible subsets.*

- `Manager *const mpManager`

*A reference to the manager.*

- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*

- `std::vector<Model> mAllModels`

*Stores all satisfying assignments.*

- `bool mModelComputed`

*True, if the model has already been computed.*

- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*

- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*

- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*

- `std::vector<TheoryPropagation> mTheoryPropagations`

- `std::atomic<Answer> mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- `Conditionals mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- `std::vector<Module*> mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- `std::vector<Module*> mAllBackends`

*The backends of this module which have been used.*

- `std::vector<Lemma> mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*

- `ModuleInput::iterator mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- `carl::FastSet<FormulaT> mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet<FormulaT> mInformedConstraints`

*Stores the position of the first constraint of which no backend has been informed about.*

- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- `unsigned mSmallerMusesCheckCounter`

*Counter used for the generation of the smt2 files to check for smaller muses.*

- `std::vector<std::size_t> mVariableCounters`

*Maps variables to the number of their occurrences.*

### 0.15.310.1 Member Typedef Documentation

**0.15.310.1.1 SettingsType** template<typename Settings >  
 typedef `Settings smtrat::MCBModule< Settings >::SettingsType`

### 0.15.310.2 Member Enumeration Documentation

**0.15.310.2.1 LemmaType** enum `smtrat::Module::LemmaType` : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.310.3 Constructor & Destructor Documentation

**0.15.310.3.1 MCBModule()** template<typename Settings >  
`smtrat::MCBModule< Settings >::MCBModule` (  
   const `ModuleInput` \* `_formula`,  
   `Conditionals` & `_conditionals`,  
   `Manager` \* `_manager` = `nullptr` )

**0.15.310.3.2 ~MCBModule()** template<typename Settings >  
`smtrat::MCBModule< Settings >::~MCBModule` ( )

### 0.15.310.4 Member Function Documentation

**0.15.310.4.1 add()** bool `smtrat::PModule::add` (  
   `ModuleInput::const_iterator` `_subformula` ) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.310.4.2 addConstraintToInform()** void `smtrat::Module::addConstraintToInform` (  
   const `FormulaT` & `_constraint` ) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                    |                        |
|--------------------|------------------------|
| <i>_constraint</i> | The constraint to add. |
|--------------------|------------------------|

Reimplemented in [smrat::SATModule< Settings >](#).

**0.15.310.4.3 addCore()** `virtual bool smrat::Module::addCore (`

`ModuleInput::const_iterator formula ) [inline], [protected], [virtual], [inherited]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>formula</i> | The sub-formula to take additionally into account. |
|----------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.310.4.4 addInformationRelevantFormula()** `void smrat::Module::addInformationRelevantFormula (`

`const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.310.4.5 addLemma()** `void smrat::Module::addLemma (`

`const FormulaT & _lemma,`

`const LemmaType & _lt = LemmaType::NORMAL,`

`const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],`

`[inherited]`

Stores a lemma being a valid formula.

**Parameters**

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.310.4.6 addOrigin()** void smtrat::Module::addOrigin (   
     ModuleInput::iterator \_formula,  
     const FormulaT & \_origin ) [inline], [protected], [inherited]  
 Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

**0.15.310.4.7 addReceivedSubformulaToPassedFormula()** std::pair<ModuleInput::iterator,bool> smtrat::Module::addReceivedSubformulaToPassedFormula (   
     ModuleInput::const\_iterator \_subformula ) [inline], [protected], [inherited]

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

**0.15.310.4.8 addSubformulaToPassedFormula() [1/3]** std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (   
     const FormulaT & \_formula ) [inline], [protected], [inherited]

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.310.4.9 addSubformulaToPassedFormula() [2/3]** std::pair<ModuleInput::iterator,bool> smtrat::Module::addSubformulaToPassedFormula (   
     const FormulaT & \_formula,  
     const FormulaT & \_origin ) [inline], [protected], [inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                 |
| _origin  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.310.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>**

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.310.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]****Returns**

All satisfying assignments, if existent.

**0.15.310.4.12 anAnswerFound() bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.310.4.13 answerFound() const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.310.4.14 appliedPreprocessing() bool smrat::PModule::appliedPreprocessing () const [inline], [inherited]****Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.310.4.15 `backendsModel()`** const `Model` & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.310.4.16 `branchAt() [1/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.310.4.17 `branchAt() [2/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.310.4.18 `branchAt() [3/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.310.4.19 `branchAt() [4/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [protected], [inherited]

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                            |                                                                                                                                                                                                                |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>   | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>     | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>        | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>      | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code> | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |

**Parameters**

|                                           |                                                                                              |
|-------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise. |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                |

**0.15.310.4.20 `check()`** `Answer smtrat::PModule::check (`

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.310.4.21 `checkCore()`** `template<typename Settings >`

```
Answer smtrat::MCBModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

**Returns**

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.310.4.22 `checkInfSubsetForMinimality()`** `void smtrat::Module::checkInfSubsetForMinimality (`

```
 std::vector< FormulaSet >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.310.4.23 `checkModel()`** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.310.4.24 `cleanModel()`** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.310.4.25 `clearLemmas()`** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.310.4.26 `clearModel()`** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.310.4.27 `clearModels()`** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.310.4.28 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.310.4.29 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.310.4.30 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.310.4.31 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

**0.15.310.4.32 constraintsToInform()** const carl::FastSet<[FormulaT](#)>& smrat::Module::constraintsToInform( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.310.4.33 currentlySatisfied()** virtual unsigned smrat::Module::currentlySatisfied( const [FormulaT](#) & ) const [inline], [virtual], [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.310.4.34 currentlySatisfiedByBackend()** unsigned smrat::Module::currentlySatisfiedByBackend( const [FormulaT](#) & \_formula ) const [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.310.4.35 deinform()** void smrat::Module::deinform( const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.310.4.36 deinformCore()** virtual void smrat::Module::deinformCore( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.310.4.37 determine\_smallest\_origin()** size\_t smrat::Module::determine\_smallest\_origin( const std::vector< [FormulaT](#) > & origins ) const [protected], [inherited]

## Parameters

|                       |                              |
|-----------------------|------------------------------|
| <code>_origins</code> | A vector of sets of origins. |
|-----------------------|------------------------------|

## Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.310.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

```
ModuleInput::iterator _subformula,
bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

## Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

## Returns

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.310.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.310.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`

```
const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

## Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

## Returns

**0.15.310.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.310.4.42 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.310.4.43 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.310.4.44 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.310.4.45 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.310.4.46 getBackendsModel()** `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.310.4.47 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.310.4.48 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (`

`const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.310.4.49 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.310.4.50 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.310.4.51 `getOrigins() [1/3]`** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulaSetT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.310.4.52 `getOrigins() [2/3]`** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulasT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.310.4.53 `getOrigins() [3/3]`** `const FormulaT& smtrat::Module::getOrigins (`

```
ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.310.4.54 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.310.4.55 hasLemmas()** `bool smrat::Module::hasLemmas () [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.310.4.56 hasValidInfeasibleSubset()** `bool smrat::Module::hasValidInfeasibleSubset () const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.310.4.57 id()** `std::size_t smrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.310.4.58 infeasibleSubsets()** `const std::vector<FormulaSetT>& smrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.310.4.59 inform()** `bool smrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.310.4.60 informBackends()** `void smrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.310.4.61 informCore()** `virtual bool smrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

`false`, if it can be easily decided whether the given constraint is inconsistent; `true`, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

#### 0.15.310.4.62 informedConstraints()

```
const carl::FastSet<FormulaT>& smrat::Module::informed<→
Constraints () const [inline], [inherited]
```

#### Returns

The position of the first constraint of which no backend has been informed about.

#### 0.15.310.4.63 init()

```
void smrat::Module::init () [virtual], [inherited]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

#### 0.15.310.4.64 is\_minimizing()

```
bool smrat::Module::is_minimizing () const [inline], [inherited]
```

#### 0.15.310.4.65 isLemmaLevel()

```
bool smrat::Module::isLemmaLevel (
```

`LemmaLevel level`) [protected], [inherited]

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

#### 0.15.310.4.66 isPreprocessor()

```
bool smrat::PModule::isPreprocessor () const [inline], [inherited]
```

#### Returns

`true`, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.310.4.67 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.310.4.68 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & [\\_vecSetA](#), const std::vector< [FormulaT](#) > & [\\_vecSetB](#) ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                          |                                  |
|--------------------------|----------------------------------|
| <a href="#">_vecSetA</a> | A vector of sets of constraints. |
| <a href="#">_vecSetB</a> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.310.4.69 model()** const [Model](#)& smtrat::Module::model ( ) const [inline], [inherited]

**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.310.4.70 modelsDisjoint()** bool smtrat::Module::modelsDisjoint ( const [Model](#) & [\\_modelA](#), const [Model](#) & [\\_modelB](#) ) [static], [protected], [inherited]

Checks whether there is no variable assigned by both models.

**Parameters**

|                         |                                |
|-------------------------|--------------------------------|
| <a href="#">_modelA</a> | The first model to check for.  |
| <a href="#">_modelB</a> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.310.4.71 moduleName()** virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]

**Returns**

The name of the given module type as name.

Reimplemented in `smrat::VSModule< Settings >`, `smrat::UnionFindModule< Settings >`, `smrat::UFCegarModule< Settings >`, `smrat::STropModule< Settings >`, `smrat::SplitSOSModule< Settings >`, `smrat::PBPPModule< Settings >`, `smrat::PBGaussModule< Settings >`, `smrat::NRAILModule< Settings >`, `smrat::NewCADModule< Settings >`, `smrat::LVEModule< Settings >`, `smrat::LRAModule< Settings >`, `smrat::LRAModule< LRASettingsICP >`, `smrat::LRAModule< LRASettings1 >`, `smrat::IntEqModule< Settings >`, `smrat::IntBlastModule< Settings >`, `smrat::IncWidthModule< Settings >`, `smrat::ICPModule< Settings >`, `smrat::ICPModule< ICPSettings4 >`, `smrat::ICPModule< ICPSettings1 >`, `smrat::GBModule< Settings >`, `smrat::FouMoModule< Settings >`, `smrat::EQPreprocessingModule< Settings >`, `smrat::EQModule< Settings >`, `smrat::CurryModule< Settings >`, `smrat::CubeLIAModule< Settings >`, `smrat::CSplitModule< Settings >`, `smrat::BVMModule< Settings >`, and `smrat::BEModule< Settings >`.

**0.15.310.4.72 `objective()`** `carl::Variable smrat::Module::objective ( ) const [inline], [inherited]`**0.15.310.4.73 `originInReceivedFormula()`** `bool smrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`**0.15.310.4.74 `passedFormulaBegin()`** `ModuleInput::iterator smrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.310.4.75 `passedFormulaEnd()`** `ModuleInput::iterator smrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula.

**0.15.310.4.76 `pPassedFormula()`** `const ModuleInput* smrat::Module::pPassedFormula ( ) const [inline], [inherited]`**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.310.4.77 `pReceivedFormula()`** `const ModuleInput* smrat::Module::pReceivedFormula ( ) const [inline], [inherited]`**Returns**

A pointer to the formula passed to this module.

**0.15.310.4.78 `print()`** `void smrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.310.4.79 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.310.4.80 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.310.4.81 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.310.4.82 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.310.4.83 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.310.4.84 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.310.4.85 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline], [inherited]
```

Notifies that the received formulas has been checked.

```
0.15.310.4.86 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs<-
InfeasibleSubset (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.310.4.87 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.310.4.88 remove() void smtrat::PModule::remove (
```

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

```
0.15.310.4.89 removeCore() virtual void smtrat::Module::removeCore (
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.310.4.90 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.310.4.91 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins (`

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.310.4.92 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const`

[inline], [inherited]

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.310.4.93 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const`

[inline], [inherited]

#### Returns

A reference to the formula passed to this module.

**0.15.310.4.94 runBackends() [1/2]** `virtual Answer smrat::PModule::runBackends ( ) [inline], [virtual], [inherited]`

Reimplemented from [smrat::Module](#).

**0.15.310.4.95 runBackends() [2/2]** `Answer smrat::PModule::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.310.4.96 setId()** `void smtrat::Module::setId ( std::size_t _id ) [inline], [inherited]`  
 Sets this modules unique ID to identify itself.

**Parameters**

|              |                                    |
|--------------|------------------------------------|
| $\leftarrow$ | The id to set this module's id to. |
| $\_id$       |                                    |

**0.15.310.4.97 setThreadPriority()** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`  
 Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| $\_threadPriority$ | The priority to set this module's thread priority to. |
|--------------------|-------------------------------------------------------|

**0.15.310.4.98 solverState()** `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.310.4.99 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                       |                                                  |
|-----------------------|--------------------------------------------------|
| $\_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|-----------------------|--------------------------------------------------|

**0.15.310.4.100 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.310.4.101 updateAllModels()** void smrat::Module::updateAllModels () [virtual], [inherited]  
Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in [smrat::SATModule< Settings >](#).

**0.15.310.4.102 updateLemmas()** void smrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.310.4.103 updateModel()** template<typename Settings >  
void [smrat::MCBModule< Settings >](#)::updateModel () const [virtual]  
Updates the model, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented from [smrat::Module](#).

**0.15.310.4.104 usedBackends()** const std::vector<[Module\\*](#)>& smrat::Module::usedBackends ()  
const [inline], [inherited]

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.310.5 Field Documentation

**0.15.310.5.1 mAllBackends** std::vector<[Module\\*](#)> smrat::Module::mAllBackends [protected], [inherited]  
The backends of this module which have been used.

**0.15.310.5.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.310.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.310.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.310.5.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.310.5.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.310.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass` [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.310.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.310.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer` [protected], [inherited]

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.310.5.10 mFullCheck** `bool smrat::Module::mFullCheck` [protected], [inherited]

false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.310.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smrat::Module::mInfeasibleSubsets`

[protected], [inherited]

Stores the infeasible subsets.

**0.15.310.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints`

[protected], [inherited]

Stores the position of the first constraint of which no backend has been informed about.

**0.15.310.5.13 mLastBranches** `std::vector< Branching > smrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.310.5.14 mLemmas** `std::vector<Lemma> smrat::Module::mLemmas` [protected], [inherited]

Stores the lemmas being valid formulas this module or its backends made.

**0.15.310.5.15 mModel** `Model smrat::Module::mModel` [mutable], [protected], [inherited]

Stores the assignment of the current satisfiable result, if existent.

**0.15.310.5.16 mModelComputed** `bool smrat::Module::mModelComputed` [mutable], [protected], [inherited]

True, if the model has already been computed.

**0.15.310.5.17 mNumOfBranchVarsToStore** `std::size_t smrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.310.5.18 mObjectiveVariable** carl::Variable smrat::Module::mObjectiveVariable [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.310.5.19 mOldSplittingVariables** std::vector< [FormulaT](#) > smrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.310.5.20 mpManager** Manager\* const smrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.310.5.21 mSmallerMusesCheckCounter** unsigned smrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.310.5.22 mSolverState** std::atomic<[Answer](#)> smrat::Module::mSolverState [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.310.5.23 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.310.5.24 mUsedBackends** std::vector<[Module](#)\*> smrat::Module::mUsedBackends [protected], [inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.310.5.25 mVariableCounters** std::vector<std::size\_t> smrat::Module::mVariableCounters [protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.311 smrat::mcsat::MCASATBackend< Settings > Class Template Reference

```
#include <BaseBackend.h>
```

### Public Member Functions

- void [pushConstraint](#) (const [FormulaT](#) &f)
- void [popConstraint](#) (const [FormulaT](#) &f)
- void [pushAssignment](#) (carl::Variable v, const [ModelValue](#) &mv, const [FormulaT](#) &f)
- void [popAssignment](#) (carl::Variable v)
- const auto & [getModel](#) () const
- const auto & [getTrail](#) () const
- void [initVariables](#) (const carl::Variables &variables)
- const auto & [variables](#) () const
- const auto & [assignedVariables](#) () const
- [AssignmentOrConflict](#) [findAssignment](#) (carl::Variable var) const
- [AssignmentOrConflict](#) [isInfeasible](#) (carl::Variable var, const [FormulaT](#) &f)
- [Explanation](#) [explain](#) (carl::Variable var, const [FormulasT](#) &reason, bool force\_use\_core=false) const

- `Explanation explain (carl::Variable var, const FormulaT &f, const FormulasT &reason)`
- `bool isActive (const FormulaT &f) const`

### 0.15.311.1 Member Function Documentation

**0.15.311.1.1 assignedVariables()** template<typename Settings >  
const auto& smtrat::mcsat::MCSATBackend< Settings >::assignedVariables ( ) const [inline]

**0.15.311.1.2 explain() [1/2]** template<typename Settings >  
Explanation smtrat::mcsat::MCSATBackend< Settings >::explain (  
 carl::Variable var,  
 const FormulasT & reason,  
 bool force\_use\_core = false ) const [inline]

**0.15.311.1.3 explain() [2/2]** template<typename Settings >  
Explanation smtrat::mcsat::MCSATBackend< Settings >::explain (  
 carl::Variable var,  
 const FormulaT & f,  
 const FormulasT & reason ) [inline]

**0.15.311.1.4 findAssignment()** template<typename Settings >  
AssignmentOrConflict smtrat::mcsat::MCSATBackend< Settings >::findAssignment (  
 carl::Variable var ) const [inline]

**0.15.311.1.5 getModel()** template<typename Settings >  
const auto& smtrat::mcsat::MCSATBackend< Settings >::getModel ( ) const [inline]

**0.15.311.1.6 getTrail()** template<typename Settings >  
const auto& smtrat::mcsat::MCSATBackend< Settings >::getTrail ( ) const [inline]

**0.15.311.1.7 initVariables()** template<typename Settings >  
void smtrat::mcsat::MCSATBackend< Settings >::initVariables (  
 const carl::Variables & variables ) [inline]

**0.15.311.1.8 isActive()** template<typename Settings >  
bool smtrat::mcsat::MCSATBackend< Settings >::isActive (  
 const FormulaT & f ) const [inline]

**0.15.311.1.9 isInfeasible()** template<typename Settings >  
AssignmentOrConflict smtrat::mcsat::MCSATBackend< Settings >::isInfeasible (  
 carl::Variable var,  
 const FormulaT & f ) [inline]

**0.15.311.1.10 popAssignment()** template<typename Settings >  
void smtrat::mcsat::MCSATBackend< Settings >::popAssignment ( carl::Variable v ) [inline]

**0.15.311.1.11 popConstraint()** template<typename Settings >  
void smtrat::mcsat::MCSATBackend< Settings >::popConstraint ( const FormulaT & f ) [inline]

**0.15.311.1.12 pushAssignment()** template<typename Settings >  
void smtrat::mcsat::MCSATBackend< Settings >::pushAssignment ( carl::Variable v, const ModelValue & mv, const FormulaT & f ) [inline]

**0.15.311.1.13 pushConstraint()** template<typename Settings >  
void smtrat::mcsat::MCSATBackend< Settings >::pushConstraint ( const FormulaT & f ) [inline]

**0.15.311.1.14 variables()** template<typename Settings >  
const auto& smtrat::mcsat::MCSATBackend< Settings >::variables () const [inline]

## 0.15.312 smtrat::mcsat::MCSATMixin< Settings > Class Template Reference

#include <MCSATMixin.h>

### Public Member Functions

- template<typename BaseModule >  
**MCSATMixin** (BaseModule &baseModule)
- std::size\_t **level** () const
- const **Model** & **model** () const
- const std::vector< Minisat::Var > & **undecidedBooleanVariables** () const
- bool **isAssignedTheoryVariable** (const carl::Variable &var) const
- bool **theoryAssignmentComplete** () const
- const **TheoryLevel** & **get** (std::size\_t **level**) const
  - Returns the respective theory level.*
- const **TheoryLevel** & **current** () const
  - Returns the current theory level.*
- **TheoryLevel** & **current** ()
- carl::Variable **variable** (std::size\_t **level**) const
  - Retrieve already decided theory variables.*
- void **doBooleanAssignment** (Minisat::Lit lit)
  - Add a new constraint.*
- void **undoBooleanAssignment** (Minisat::Lit lit)
  - Remove the last constraint.*
- std::size\_t **addBooleanVariable** (Minisat::Var **variable**)
  - Add a variable, return the level it was inserted on.*
- **Minisat::Var** **topSemanticPropagation** ()
  - Getter for semantic propagations.*
- bool **isFormulaUnivariate** (const **FormulaT** &formula, std::size\_t **level**) const

*Checks whether the given formula is univariate on the given level.*

- void `pushTheoryDecision` (const `FormulaT` &assignment, `Minisat::Lit` decisionLiteral)
 

*Push a theory decision.*
- bool `backtrackTo` (`Minisat::Lit` literal)
 

*Backtracks to the theory decision represented by the given literal.*
- `Minisat::Ibool evaluateLiteral` (`Minisat::Lit` lit) const
 

*Evaluate a literal in the theory, set lastReason to last theory decision involved.*
- std::pair< bool, std::optional< `Explanation` > > `isBooleanDecisionFeasible` (`Minisat::Lit` lit, bool always\_← explain=false)
- std::pair< boost::tribool, std::optional< `Explanation` > > `propagateBooleanDomain` (`Minisat::Lit` lit)
- std::optional< `Explanation` > `isFeasible` (const `carl::Variable` &var)
- std::variant< `Explanation`, `FormulasT` > `makeTheoryDecision` (const `carl::Variable` &var)
- bool `is_consistent` ()
 

*Checks if any inconsistency was detected.*
- bool `is_consistent` (`Minisat::Var` v)
- std::optional< `Explanation` > `explainInconsistency` ()
 

*Checks if the trail is consistent after the assignment on the current level.*
- `Explanation explainTheoryPropagation` (`Minisat::Lit` literal)
- template<typename Constraints>
 `void initVariables` (const `Constraints` &c)
- bool `isTheoryVar` (`Minisat::Var` v) const
- const `carl::Variable` & `carlVar` (`Minisat::Var` v) const
- `Minisat::Var minisatVar` (const `carl::Variable` &v) const
- std::vector< `Minisat::Var` > `theoryVarAbstractions` () const
- bool `hasUnassignedDep` (`Minisat::Var` v) const
- std::size\_t `theoryLevel` (`Minisat::Var` var) const
- std::size\_t `computeTheoryLevel` (`Minisat::Var` var) const
- std::size\_t `computeTheoryLevel` (const `FormulaT` &f) const
- `Minisat::Lit getDecisionLiteral` (`Minisat::Var` var) const
- int `assignedAtTraillIndex` (`Minisat::Var` variable) const
- int `decisionLevel` (`Minisat::Var` v) const
- bool `fullConsistencyCheck` () const
- std::size\_t `maxDegree` (const `Minisat::Var` &var)
- std::vector< `Minisat::Var` > `theoryVars` (const `Minisat::Var` &var)
- void `printClause` (std::ostream &os, `Minisat::CRef` clause) const
 

*Prints a single clause.*

## Friends

- template<typename Sett>
 `std::ostream & operator<<` (`std::ostream &os, const MCSATMixin< Sett > &mcm)`

### 0.15.312.1 Constructor & Destructor Documentation

```
0.15.312.1.1 MCSATMixin() template<typename Settings >
template<typename BaseModule >
smrat::mcsat::MCSATMixin< Settings >::MCSATMixin (
 BaseModule & baseModule) [inline], [explicit]
```

### 0.15.312.2 Member Function Documentation

**0.15.312.2.1 addBooleanVariable()** template<typename Settings >  
std::size\_t smtrat::mcsat::MCSATMixin< Settings >::addBooleanVariable (  
    Minisat::Var variable )

Add a variable, return the level it was inserted on.

**0.15.312.2.2 assignedAtTrailIndex()** template<typename Settings >  
int smtrat::mcsat::MCSATMixin< Settings >::assignedAtTrailIndex (  
    Minisat::Var variable ) const [inline]

**0.15.312.2.3 backtrackTo()** template<typename Settings >  
bool smtrat::mcsat::MCSATMixin< Settings >::backtrackTo (  
    Minisat::Lit literal )

Backtracks to the theory decision represented by the given literal.

**0.15.312.2.4 carlVar()** template<typename Settings >  
const carl::Variable& smtrat::mcsat::MCSATMixin< Settings >::carlVar (  
    Minisat::Var v ) const [inline]

**0.15.312.2.5 computeTheoryLevel() [1/2]** template<typename Settings >  
std::size\_t smtrat::mcsat::MCSATMixin< Settings >::computeTheoryLevel (  
    const FormulaT & f ) const [inline]

**0.15.312.2.6 computeTheoryLevel() [2/2]** template<typename Settings >  
std::size\_t smtrat::mcsat::MCSATMixin< Settings >::computeTheoryLevel (  
    Minisat::Var var ) const [inline]

**0.15.312.2.7 current() [1/2]** template<typename Settings >  
TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::current () [inline]

**0.15.312.2.8 current() [2/2]** template<typename Settings >  
const TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::current () const [inline]  
Returns the current theory level.

**0.15.312.2.9 decisionLevel()** template<typename Settings >  
int smtrat::mcsat::MCSATMixin< Settings >::decisionLevel (  
    Minisat::Var v ) const [inline]

**0.15.312.2.10 doBooleanAssignment()** template<typename Settings >  
void smtrat::mcsat::MCSATMixin< Settings >::doBooleanAssignment (  
    Minisat::Lit lit ) [inline]

Add a new constraint.

**0.15.312.2.11 evaluateLiteral()** template<typename Settings >  
Minisat::lbool smtrat::mcsat::MCSATMixin< Settings >::evaluateLiteral (  
    Minisat::Lit lit ) const

Evaluate a literal in the theory, set lastReason to last theory decision involved.

---

**0.15.312.2.12 `explainInconsistency()`** template<typename Settings >  
`std::optional<Explanation> smtrat::mcsat::MCSATMixin< Settings >::explainInconsistency ( )`  
`[inline]`

Checks if the trail is consistent after the assignment on the current level.

The trail must be consistent on the previous level.

Returns std::nullopt if consistent and an explanation.

**0.15.312.2.13 `explainTheoryPropagation()`** template<typename Settings >  
`Explanation smtrat::mcsat::MCSATMixin< Settings >::explainTheoryPropagation (`  
`Minisat::Lit literal ) [inline]`

**0.15.312.2.14 `fullConsistencyCheck()`** template<typename Settings >  
`bool smtrat::mcsat::MCSATMixin< Settings >::fullConsistencyCheck ( ) const [inline]`

**0.15.312.2.15 `get()`** template<typename Settings >  
`const TheoryLevel& smtrat::mcsat::MCSATMixin< Settings >::get (`  
`std::size_t level ) const [inline]`

Returns the respective theory level.

**0.15.312.2.16 `getDecisionLiteral()`** template<typename Settings >  
`Minisat::Lit smtrat::mcsat::MCSATMixin< Settings >::getDecisionLiteral (`  
`Minisat::Var var ) const [inline]`

**0.15.312.2.17 `hasUnassignedDep()`** template<typename Settings >  
`bool smtrat::mcsat::MCSATMixin< Settings >::hasUnassignedDep (`  
`Minisat::Var v ) const [inline]`

**0.15.312.2.18 `initVariables()`** template<typename Settings >  
`template<typename Constraints >`  
`void smtrat::mcsat::MCSATMixin< Settings >::initVariables (`  
`const Constraints & c ) [inline]`

**0.15.312.2.19 `is_consistent() [1/2]`** template<typename Settings >  
`bool smtrat::mcsat::MCSATMixin< Settings >::is_consistent ( ) [inline]`  
 Checks if any inconsistency was detected.

**0.15.312.2.20 `is_consistent() [2/2]`** template<typename Settings >  
`bool smtrat::mcsat::MCSATMixin< Settings >::is_consistent (`  
`Minisat::Var v ) [inline]`

**0.15.312.2.21 `isAssignedTheoryVariable()`** template<typename Settings >  
`bool smtrat::mcsat::MCSATMixin< Settings >::isAssignedTheoryVariable (`  
`const carl::Variable & var ) const [inline]`

```
0.15.312.2.22 isBooleanDecisionFeasible() template<typename Settings >
std::pair<bool, std::optional<Explanation> > smtrat::mcsat::MCSATMixin< Settings >::isBooleanDecisionFeasible (
 Minisat::Lit lit,
 bool always_explain = false)
```

```
0.15.312.2.23 isFeasible() template<typename Settings >
std::optional<Explanation> smtrat::mcsat::MCSATMixin< Settings >::isFeasible (
 const carl::Variable & var) [inline]
```

```
0.15.312.2.24 isFormulaUnivariate() template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::isFormulaUnivariate (
 const FormulaT & formula,
 std::size_t level) const
```

Checks whether the given formula is univariate on the given level.

```
0.15.312.2.25 isTheoryVar() template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::isTheoryVar (
 Minisat::Var v) const [inline]
```

```
0.15.312.2.26 level() template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::level () const [inline]
```

```
0.15.312.2.27 makeTheoryDecision() template<typename Settings >
std::variant<Explanation, FormulasT> smtrat::mcsat::MCSATMixin< Settings >::makeTheoryDecision (
 const carl::Variable & var) [inline]
```

```
0.15.312.2.28 maxDegree() template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::maxDegree (
 const Minisat::Var & var) [inline]
```

```
0.15.312.2.29 minisatVar() template<typename Settings >
Minisat::Var smtrat::mcsat::MCSATMixin< Settings >::minisatVar (
 const carl::Variable & v) const [inline]
```

```
0.15.312.2.30 model() template<typename Settings >
const Model& smtrat::mcsat::MCSATMixin< Settings >::model () const [inline]
```

```
0.15.312.2.31 printClause() template<typename Settings >
void smtrat::mcsat::MCSATMixin< Settings >::printClause (
 std::ostream & os,
 Minisat::CRef clause) const
```

Prints a single clause.

```
0.15.312.2.32 propagateBooleanDomain() template<typename Settings >
std::pair<boost::tribool, std::optional<Explanation> > smtrat::mcsat::MCSATMixin< Settings
>::propagateBooleanDomain (
 Minisat::Lit lit)
```

```
0.15.312.2.33 pushTheoryDecision() template<typename Settings >
void smtrat::mcsat::MCSATMixin< Settings >::pushTheoryDecision (
 const FormulaT & assignment,
 Minisat::Lit decisionLiteral)
```

Push a theory decision.

```
0.15.312.2.34 theoryAssignmentComplete() template<typename Settings >
bool smtrat::mcsat::MCSATMixin< Settings >::theoryAssignmentComplete () const [inline]
```

```
0.15.312.2.35 theoryLevel() template<typename Settings >
std::size_t smtrat::mcsat::MCSATMixin< Settings >::theoryLevel (
 Minisat::Var var) const [inline]
```

```
0.15.312.2.36 theoryVarAbstractions() template<typename Settings >
std::vector<Minisat::Var> smtrat::mcsat::MCSATMixin< Settings >::theoryVarAbstractions ()
const [inline]
```

```
0.15.312.2.37 theoryVars() template<typename Settings >
std::vector<Minisat::Var> smtrat::mcsat::MCSATMixin< Settings >::theoryVars (
 const Minisat::Var & var) [inline]
```

```
0.15.312.2.38 topSemanticPropagation() template<typename Settings >
Minisat::Var smtrat::mcsat::MCSATMixin< Settings >::topSemanticPropagation () [inline]
Getter for semantic propagations.
```

```
0.15.312.2.39 undecidedBooleanVariables() template<typename Settings >
const std::vector<Minisat::Var>& smtrat::mcsat::MCSATMixin< Settings >::undecidedBooleanVariables () const [inline]
```

```
0.15.312.2.40 undoBooleanAssignment() template<typename Settings >
void smtrat::mcsat::MCSATMixin< Settings >::undoBooleanAssignment (
 Minisat::Lit lit) [inline]
```

Remove the last constraint.

f must be the same as the one passed to the last call of pushConstraint().

```
0.15.312.2.41 variable() template<typename Settings >
carl::Variable smtrat::mcsat::MCSATMixin< Settings >::variable (
 std::size_t level) const [inline]
```

Retrieve already decided theory variables.

### 0.15.312.3 Friends And Related Function Documentation

```
0.15.312.3.1 operator<< template<typename Settings >
template<typename Sett >
std::ostream& operator<< (
 std::ostream & os,
 const MCSATMixin< Sett > & mcm) [friend]
```

## 0.15.313 smtrat::expression::simplifier::MergeSimplifier Struct Reference

```
#include <MergeSimplifier.h>
```

### Public Member Functions

- const ExpressionContent \* simplify (const NaryExpression &expr) const
- const ExpressionContent \* operator() (const carl::Variable &expr) const
- const ExpressionContent \* operator() (const ITEExpression &expr) const
- const ExpressionContent \* operator() (const QuantifierExpression &expr) const
- const ExpressionContent \* operator() (const UnaryExpression &expr) const
- const ExpressionContent \* operator() (const BinaryExpression &expr) const
- const ExpressionContent \* operator() (const NaryExpression &expr) const
- const ExpressionContent \* operator() (const ExpressionContent \*\_ec) const

### Protected Member Functions

- virtual const ExpressionContent \* simplify (const carl::Variable &) const
- virtual const ExpressionContent \* simplify (const ITEExpression &) const
- virtual const ExpressionContent \* simplify (const QuantifierExpression &) const
- virtual const ExpressionContent \* simplify (const UnaryExpression &) const
- virtual const ExpressionContent \* simplify (const BinaryExpression &) const

### 0.15.313.1 Member Function Documentation

```
0.15.313.1.1 operator() [1/7] const ExpressionContent* smtrat::expression::simplifier::Base<→
Simplifier::operator() (
 const BinaryExpression & expr) const [inline], [inherited]
```

```
0.15.313.1.2 operator() [2/7] const ExpressionContent* smtrat::expression::simplifier::Base<→
Simplifier::operator() (
 const carl::Variable & expr) const [inline], [inherited]
```

```
0.15.313.1.3 operator() [3/7] const ExpressionContent* smtrat::expression::simplifier::Base<→
Simplifier::operator() (
 const ExpressionContent * _ec) const [inline], [inherited]
```

```
0.15.313.1.4 operator() [4/7] const ExpressionContent* smtrat::expression::simplifier::Base<→
Simplifier::operator() (
 const ITEExpression & expr) const [inline], [inherited]
```

```
0.15.313.1.5 operator() [5/7] const ExpressionContent* smtrat::expression::simplifier::Base<→
Simplifier::operator() (
 const NaryExpression & expr) const [inline], [inherited]
```

**0.15.313.1.6 operator() [6/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (const `QuantifierExpression` & `expr`) const [inline], [inherited]

**0.15.313.1.7 operator() [7/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (const `UnaryExpression` & `expr`) const [inline], [inherited]

**0.15.313.1.8 simplify() [1/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (const `BinaryExpression` & ) const [inline], [protected], [virtual], [inherited]

**0.15.313.1.9 simplify() [2/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (const `carl::Variable` & ) const [inline], [protected], [virtual], [inherited]

**0.15.313.1.10 simplify() [3/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (const `ITEExpression` & ) const [inline], [protected], [virtual], [inherited]

**0.15.313.1.11 simplify() [4/6]** const `ExpressionContent*` smtrat::expression::simplifier::Merge<→ Simplifier::simplify (const `NaryExpression` & `expr`) const [inline], [virtual]  
Reimplemented from `smtrat::expression::simplifier::BaseSimplifier`.

**0.15.313.1.12 simplify() [5/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (const `QuantifierExpression` & ) const [inline], [protected], [virtual], [inherited]

**0.15.313.1.13 simplify() [6/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (const `UnaryExpression` & ) const [inline], [protected], [virtual], [inherited]  
Reimplemented in `smtrat::expression::simplifier::NegationSimplifier`.

## 0.15.314 benchmax::Minisat Class Reference

Tool wrapper for a `Minisat` solver.

```
#include <Minisat.h>
```

### Public Member Functions

- `Minisat` (const `fs::path` &`binary`, const `std::string` &`arguments`)
 

*Create tool.*
- `virtual bool canHandle` (const `fs::path` &`path`) const override
 

*Only handles .opb files.*
- `virtual void additionalResults` (const `fs::path` &, `BenchmarkResult` &`result`) const override
 

*Parse results from stdout.*
- `std::string name ()` const

- Common name of this tool.*
- `fs::path binary () const`  
*Full path to the binary.*
  - `const std::map< std::string, std::string > & attributes () const`  
*A set of attributes, for example compilation options.*
  - `std::vector< std::string > resolveDependencies () const`  
*Get dependencies of binary required to run it (via ldd)*
  - `std::size_t attributeHash () const`  
*Hash of the attributes.*
  - `virtual std::string getCommandline (const std::string &file) const`  
*Compose commandline for this tool and the given input file.*
  - `virtual std::string getCommandline (const std::string &file, const std::string &localBinary) const`  
*Compose commandline for this tool with another binary name and the given input file.*
  - `virtual std::optional< std::string > parseCommandline (const std::string &cmdline) const`  
*Compose commandline for this tool and the given input file.*

## Protected Attributes

- `std::string mName`  
*(Non-unique) identifier for the tool.*
- `fs::path mBinary`  
*Path to the binary.*
- `std::string mArguments`  
*Command line arguments that should be passed to the binary.*
- `std::map< std::string, std::string > mAttributes`  
*Attributes of the tool obtained by introspection of the binary.*

### 0.15.314.1 Detailed Description

Tool wrapper for a [Minisat](#) solver.

### 0.15.314.2 Constructor & Destructor Documentation

**0.15.314.2.1 Minisat()** `benchmax::Minisat::Minisat (`  
    `const fs::path & binary,`  
    `const std::string & arguments ) [inline]`

Create tool.

### 0.15.314.3 Member Function Documentation

**0.15.314.3.1 additionalResults()** `virtual void benchmax::Minisat::additionalResults (`  
    `const fs::path & ,`  
    `BenchmarkResult & result ) const [inline], [override], [virtual]`

Parse results from stdout.

Reimplemented from [benchmax::Tool](#).

**0.15.314.3.2 attributeHash()** `std::size_t benchmax::Tool::attributeHash ( ) const [inline], [inherited]`

Hash of the attributes.

**0.15.314.3.3 attributes()** const std::map<std::string, std::string>& benchmax::Tool::attributes () const [inline], [inherited]  
A set of attributes, for example compilation options.

**0.15.314.3.4 binary()** fs::path benchmax::Tool::binary () const [inline], [inherited]  
Full path to the binary.

**0.15.314.3.5 canHandle()** virtual bool benchmax::Minisat::canHandle (const fs::path & path) const [inline], [override], [virtual]  
Only handles .opb files.  
Reimplemented from [benchmax::Tool](#).

**0.15.314.3.6 getCommandLine() [1/2]** virtual std::string benchmax::Tool::getCommandLine (const std::string & file) const [inline], [virtual], [inherited]  
Compose commandline for this tool and the given input file.

**0.15.314.3.7 getCommandLine() [2/2]** virtual std::string benchmax::Tool::getCommandLine (const std::string & file, const std::string & localBinary) const [inline], [virtual], [inherited]  
Compose commandline for this tool with another binary name and the given input file.

**0.15.314.3.8 name()** std::string benchmax::Tool::name () const [inline], [inherited]  
Common name of this tool.

**0.15.314.3.9 parseCommandLine()** virtual std::optional<std::string> benchmax::Tool::parseCommandline (const std::string & cmdline) const [inline], [virtual], [inherited]  
Compose commandline for this tool and the given input file.

**0.15.314.3.10 resolveDependencies()** std::vector<std::string> benchmax::Tool::resolveDependencies () const [inline], [inherited]  
Get dependencies of binary required to run it (via ldd)

## 0.15.314.4 Field Documentation

**0.15.314.4.1 mArguments** std::string benchmax::Tool::mArguments [protected], [inherited]  
Command line arguments that should be passed to the binary.

**0.15.314.4.2 mAttributes** std::map<std::string, std::string> benchmax::Tool::mAttributes [protected], [inherited]  
Attributes of the tool obtained by introspection of the binary.

**0.15.314.4.3 mBinary** fs::path benchmax::Tool::mBinary [protected], [inherited]  
Path to the binary.

**0.15.314.4.4 mName** std::string benchmax::Tool::mName [protected], [inherited]  
(Non-unique) identifier for the tool.

## 0.15.315 benchmax::Minisatp Class Reference

Tool wrapper for the [Minisatp](#) solver for pseudo-Boolean problems.

```
#include <Minisatp.h>
```

### Public Member Functions

- **Minisatp** (const fs::path &[binary](#), const std::string &[arguments](#))  
*Create tool, add "-v0" to arguments.*
- virtual bool [canHandle](#) (const fs::path &[path](#)) const override  
*Only handles .opb files.*
- virtual void [additionalResults](#) (const fs::path &, [BenchmarkResult](#) &[result](#)) const override  
*Parse results from stdout.*
- std::string [name](#) () const  
*Common name of this tool.*
- fs::path [binary](#) () const  
*Full path to the binary.*
- const std::map< std::string, std::string > &[attributes](#) () const  
*A set of attributes, for example compilation options.*
- std::vector< std::string > [resolveDependencies](#) () const  
*Get dependencies of binary required to run it (via ldd)*
- std::size\_t [attributeHash](#) () const  
*Hash of the attributes.*
- virtual std::string [getCommandline](#) (const std::string &[file](#)) const  
*Compose commandline for this tool and the given input file.*
- virtual std::string [getCommandline](#) (const std::string &[file](#), const std::string &[localBinary](#)) const  
*Compose commandline for this tool with another binary name and the given input file.*
- virtual std::optional< std::string > [parseCommandline](#) (const std::string &[cmdline](#)) const  
*Compose commandline for this tool and the given input file.*

### Protected Attributes

- std::string [mName](#)  
(Non-unique) identifier for the tool.
- fs::path [mBinary](#)  
Path to the binary.
- std::string [mArguments](#)  
Command line arguments that should be passed to the binary.
- std::map< std::string, std::string > [mAttributes](#)  
Attributes of the tool obtained by introspection of the binary.

### 0.15.315.1 Detailed Description

Tool wrapper for the [Minisatp](#) solver for pseudo-Boolean problems.

### 0.15.315.2 Constructor & Destructor Documentation

```
0.15.315.2.1 Minisatp() benchmax::Minisatp::Minisatp (
 const fs::path & binary,
 const std::string & arguments) [inline]
```

Create tool, add "-v0" to arguments.

### 0.15.315.3 Member Function Documentation

```
0.15.315.3.1 additionalResults() virtual void benchmax::Minisatp::additionalResults (
 const fs::path & ,
 BenchmarkResult & result) const [inline], [override], [virtual]
```

Parse results from stdout.

Reimplemented from [benchmax::Tool](#).

```
0.15.315.3.2 attributeHash() std::size_t benchmax::Tool::attributeHash () const [inline],
[inherited]
```

Hash of the attributes.

```
0.15.315.3.3 attributes() const std::map<std::string, std::string>& benchmax::Tool::attributes (
) const [inline], [inherited]
```

A set of attributes, for example compilation options.

```
0.15.315.3.4 binary() fs::path benchmax::Tool::binary () const [inline], [inherited]
```

Full path to the binary.

```
0.15.315.3.5 canHandle() virtual bool benchmax::Minisatp::canHandle (
 const fs::path & path) const [inline], [override], [virtual]
```

Only handles .opb files.

Reimplemented from [benchmax::Tool](#).

```
0.15.315.3.6 getCommandLine() [1/2] virtual std::string benchmax::Tool::getCommandLine (
 const std::string & file) const [inline], [virtual], [inherited]
```

Compose commandline for this tool and the given input file.

```
0.15.315.3.7 getCommandLine() [2/2] virtual std::string benchmax::Tool::getCommandLine (
 const std::string & file,
 const std::string & localBinary) const [inline], [virtual], [inherited]
```

Compose commandline for this tool with another binary name and the given input file.

```
0.15.315.3.8 name() std::string benchmax::Tool::name () const [inline], [inherited]
```

Common name of this tool.

```
0.15.315.3.9 parseCommandLine() virtual std::optional<std::string> benchmax::Tool::parseCommandline (
 const std::string & cmdline) const [inline], [virtual], [inherited]
```

Compose commandline for this tool and the given input file.

**0.15.315.3.10 resolveDependencies()** std::vector<std::string> benchmax::Tool::resolveDependencies()  
() const [inline], [inherited]  
Get dependencies of binary required to run it (via ldd)

#### 0.15.315.4 Field Documentation

**0.15.315.4.1 mArguments** std::string benchmax::Tool::mArguments [protected], [inherited]  
Command line arguments that should be passed to the binary.

**0.15.315.4.2 mAttributes** std::map<std::string, std::string> benchmax::Tool::mAttributes [protected],  
[inherited]  
Attributes of the tool obtained by introspection of the binary.

**0.15.315.4.3 mBinary** fs::path benchmax::Tool::mBinary [protected], [inherited]  
Path to the binary.

**0.15.315.4.4 mName** std::string benchmax::Tool::mName [protected], [inherited]  
(Non-unique) identifier for the tool.

### 0.15.316 smtrat::mcsat::fm::MinSizeComparator Struct Reference

This heuristic chooses the explanation excluding the smallest interval.

```
#include <ConflictGenerator.h>
```

#### Public Member Functions

- bool [operator\(\)](#) (const [Bound](#) &b1, const [Bound](#) &b2) const

#### Data Fields

- bool [symmetric](#) = true

#### 0.15.316.1 Detailed Description

This heuristic chooses the explanation excluding the smallest interval.

#### 0.15.316.2 Member Function Documentation

**0.15.316.2.1 operator()** bool smtrat::mcsat::fm::MinSizeComparator::operator() (const [Bound](#) & b1, const [Bound](#) & b2) const [inline]

#### 0.15.316.3 Field Documentation

**0.15.316.3.1 symmetric** bool smtrat::mcsat::fm::MinSizeComparator::symmetric = true

### 0.15.317 smtrat::mcsat::fm::MinVarCountComparator Struct Reference

This heuristic tries to minimize the number of variables occurring in the explanation.

```
#include <ConflictGenerator.h>
```

#### Public Member Functions

- bool `operator()` (const `Bound` &`b1`, const `Bound` &`b2`) const

#### Data Fields

- bool `symmetric` = false

#### 0.15.317.1 Detailed Description

This heuristic tries to minimize the number of variables occurring in the explanation.  
It is a 2-approximation to the lowest possible number of variables in an explanation.

#### 0.15.317.2 Member Function Documentation

```
0.15.317.2.1 operator() bool smtrat::mcsat::fm::MinVarCountComparator::operator() (
 const Bound & b1,
 const Bound & b2) const [inline]
```

#### 0.15.317.3 Field Documentation

```
0.15.317.3.1 symmetric bool smtrat::mcsat::fm::MinVarCountComparator::symmetric = false
```

### 0.15.318 smtrat::cad::MISGeneration< heuristic > Class Template Reference

```
#include <MISGeneration.h>
```

#### Public Member Functions

- template<typename CAD >
 void `operator()` (const `CAD` &`cad`, std::vector< `FormulaSetT` > &`mis`)
- void `operator()` (const `CAD` &`cad`, std::vector< `FormulaSetT` > &`mis`)
- void `operator()` (const `CAD` &`cad`, std::vector< `FormulaSetT` > &`mis`)
- void `operator()` (const `CAD` &`cad`, std::vector< `FormulaSetT` > &`mis`)
- void `operator()` (const `CAD` &`cad`, std::vector< `FormulaSetT` > &`mis`)
- void `operator()` (const `CAD` &`cad`, std::vector< `FormulaSetT` > &`mis`)
- void `operator()` (const `CAD` &`cad`, std::vector< `FormulaSetT` > &`mis`)

#### 0.15.318.1 Member Function Documentation

```
0.15.318.1.1 operator() [1/7] template< MISHuristic heuristic >
template<typename CAD >
void smtrat::cad::MISGeneration< heuristic >::operator() (
 const CAD & cad,
 std::vector< FormulaSetT > & mis)
```

```
0.15.318.1.2 operator() [2/7] void smtrat::cad::MISGeneration< MISHeuristic::TRIVIAL >::operator()
(
 const CAD & cad,
 std::vector< FormulaSetT > & mis)
```

  

```
0.15.318.1.3 operator() [3/7] void smtrat::cad::MISGeneration< MISHeuristic::GREEDY >::operator()
(
 const CAD & cad,
 std::vector< FormulaSetT > & mis)
```

  

```
0.15.318.1.4 operator() [4/7] void smtrat::cad::MISGeneration< MISHeuristic::GREEDY_PRE >::operator()
(
 const CAD & cad,
 std::vector< FormulaSetT > & mis)
```

  

```
0.15.318.1.5 operator() [5/7] void smtrat::cad::MISGeneration< MISHeuristic::EXACT >::operator()
(
 const CAD & cad,
 std::vector< FormulaSetT > & mis)
```

  

```
0.15.318.1.6 operator() [6/7] void smtrat::cad::MISGeneration< MISHeuristic::HYBRID >::operator()
(
 const CAD & cad,
 std::vector< FormulaSetT > & mis)
```

  

```
0.15.318.1.7 operator() [7/7] void smtrat::cad::MISGeneration< MISHeuristic::GREEDY_WEIGHTED >::operator()
(
 const CAD & cad,
 std::vector< FormulaSetT > & mis)
```

## 0.15.319 smtrat::MixedSignEncoder Class Reference

```
#include <MixedSignEncoder.h>
```

### Public Member Functions

- `MixedSignEncoder ()`
- `bool canEncode (const ConstraintT &constraint)`
- `Rational encodingSize (const ConstraintT &constraint)`
- `std::string name ()`
- `std::optional< FormulaT > encode (const ConstraintT &constraint)`

*Encodes an arbitrary constraint.*

### Data Fields

- `std::size_t problem_size`

### Protected Member Functions

- `std::optional< FormulaT > doEncode (const ConstraintT &constraint)`
- `FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)`

### 0.15.319.1 Constructor & Destructor Documentation

**0.15.319.1.1 MixedSignEncoder()** smtrat::MixedSignEncoder::MixedSignEncoder ( ) [inline]

### 0.15.319.2 Member Function Documentation

**0.15.319.2.1 canEncode()** bool smtrat::MixedSignEncoder::canEncode ( const ConstraintT & constraint ) [virtual]  
Implements [smtrat::PseudoBoolEncoder](#).

**0.15.319.2.2 doEncode()** std::optional< [FormulaT](#) > smtrat::MixedSignEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]  
Implements [smtrat::PseudoBoolEncoder](#).

**0.15.319.2.3 encode()** std::optional< [FormulaT](#) > smtrat::PseudoBoolEncoder::encode ( const ConstraintT & constraint ) [inherited]  
Encodes an arbitrary constraint.

Returns

encoded formula

**0.15.319.2.4 encodingSize()** Rational smtrat::MixedSignEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

**0.15.319.2.5 generateVarChain()** [FormulaT](#) smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]

**0.15.319.2.6 name()** std::string smtrat::MixedSignEncoder::name ( ) [inline], [virtual]  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

### 0.15.319.3 Field Documentation

**0.15.319.3.1 problem\_size** std::size\_t smtrat::PseudoBoolEncoder::problem\_size [inherited]

**0.15.320 smtrat::cad::ModelBasedProjection< incrementality, backtracking, Settings > Class Template Reference**

#include <Projection.h>

## Public Member Functions

- `std::size_t dim () const`  
*Returns the dimension of the projection.*
- `const auto & vars () const`  
*Returns the variables used for projection.*
- `void reset ()`  
*Resets all datastructures, use the given variables from now on.*
- template<typename F>  
`void setRemoveCallback (F &&f)`  
*Sets a callback that is called whenever polynomials are removed.*
- `carl::Bitset addPolynomial (const UPoly &p, std::size_t cid, bool isBound)`  
*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- `virtual carl::Bitset addPolynomial (const UPoly &p, std::size_t cid, bool isBound)=0`  
*Adds the given polynomial to the projection.*
- `carl::Bitset addEqConstraint (const Poly &p, std::size_t cid, bool isBound)`  
*Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- `virtual carl::Bitset addEqConstraint (const UPoly &p, std::size_t cid, bool isBound)`  
*Adds the given polynomial of an equational constraint to the projection.*
- `void removePolynomial (const Poly &p, std::size_t cid, bool isBound)`  
*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- `virtual void removePolynomial (const UPoly &p, std::size_t cid, bool isBound)=0`  
*Removes the given polynomial from the projection.*
- `virtual std::size_t size (std::size_t level) const =0`
- `std::size_t size () const`
- `virtual bool empty (std::size_t level) const =0`
- `virtual bool empty ()`
- `OptionalID getPolyForLifting (std::size_t level, SampleLiftedWith &slw)`  
*Get a polynomial from this level suited for lifting.*
- `virtual bool hasPolynomialByld (std::size_t level, std::size_t id) const =0`
- `virtual const UPoly & getPolynomialByld (std::size_t level, std::size_t id) const =0`  
*Retrieves a polynomial from its id.*
- `virtual void exportAsDot (std::ostream &) const`
- `virtual Origin getOrigin (std::size_t level, std::size_t id) const`

## Protected Types

- using `Constraints = CADConstraints< Settings::backtracking >`

## Protected Member Functions

- `void callRemoveCallback (std::size_t level, const SampleLiftedWith &slw) const`
- `std::size_t getID (std::size_t level)`  
*Returns a fresh polynomial id for the given level.*
- `void freeID (std::size_t level, std::size_t id)`  
*Frees a currently used polynomial id for the given level.*
- `carl::Variable var (std::size_t level) const`  
*Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
- `bool canBePurgedByBounds (const UPoly &p) const`  
*Checks whether a polynomial can safely be ignored due to the bounds.*
- `bool isPurged (std::size_t level, std::size_t id)`

## Protected Attributes

- const `Constraints` & `mConstraints`
- std::vector< `PolyLiftingQueue< BaseProjection >` > `mLiftingQueues`  
*List of lifting queues that can be used for incremental projection.*
- `ProjectionInformation mInfo`  
*Additional info on projection, projection levels and projection polynomials.*
- `ProjectionOperator mOperator`  
*The projection operator.*
- std::function< void(std::size\_t, const `SampleLiftedWith` &) > `mRemoveCallback`  
*Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

### 0.15.320.1 Member Typedef Documentation

**0.15.320.1.1 Constraints** template<typename Settings >  
using `smtrat::cad::BaseProjection< Settings >`::`Constraints` = `CADConstraints<Settings::backtracking>`  
[protected], [inherited]

### 0.15.320.2 Member Function Documentation

**0.15.320.2.1 addEqConstraint() [1/2]** template<typename Settings >  
carl::Bitset `smtrat::cad::BaseProjection< Settings >`::`addEqConstraint` (  
  const `Poly` & `p`,  
  std::size\_t `cid`,  
  bool `isBound`) [inline], [inherited]

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.320.2.2 addEqConstraint() [2/2]** template<typename Settings >  
virtual carl::Bitset `smtrat::cad::BaseProjection< Settings >`::`addEqConstraint` (  
  const `UPoly` & `p`,  
  std::size\_t `cid`,  
  bool `isBound`) [inline], [virtual], [inherited]

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

**0.15.320.2.3 addPolynomial() [1/2]** template<typename Settings >  
carl::Bitset `smtrat::cad::BaseProjection< Settings >`::`addPolynomial` (  
  const `Poly` & `p`,  
  std::size\_t `cid`,  
  bool `isBound`) [inline], [inherited]

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.320.2.4 addPolynomial() [2/2]** template<typename Settings >  
virtual carl::Bitset `smtrat::cad::BaseProjection< Settings >`::`addPolynomial` (  
  const `UPoly` & `p`,  
  std::size\_t `cid`,  
  bool `isBound`) [pure virtual], [inherited]

Adds the given polynomial to the projection.

Implemented in `smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`, `smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`, and `smrat::cad::Projection< Incrementality::SIMPLE, Backtracking::HIDE, Settings >`.

**0.15.320.2.5 `callRemoveCallback()`** template<typename Settings>  
void `smrat::cad::BaseProjection< Settings >`::`callRemoveCallback` (  
    std::size\_t level,  
    const `SampleLiftedWith` & slw) const [inline], [protected], [inherited]

**0.15.320.2.6 `canBePurgedByBounds()`** template<typename Settings>  
bool `smrat::cad::BaseProjection< Settings >`::`canBePurgedByBounds` (  
    const `UPoly` & p) const [inline], [protected], [inherited]

Checks whether a polynomial can safely be ignored due to the bounds.

**0.15.320.2.7 `dim()`** template<typename Settings>  
std::size\_t `smrat::cad::BaseProjection< Settings >`::`dim` () const [inline], [inherited]

Returns the dimension of the projection.

**0.15.320.2.8 `empty() [1/2]`** template<typename Settings>  
virtual bool `smrat::cad::BaseProjection< Settings >`::`empty` () [inline], [virtual], [inherited]

**0.15.320.2.9 `empty() [2/2]`** template<typename Settings>  
virtual bool `smrat::cad::BaseProjection< Settings >`::`empty` (  
    std::size\_t level) const [pure virtual], [inherited]

Implemented in `smrat::qe::cad::Projection< Settings >`, `smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`, `smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::HIDE, Settings >`, `smrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

**0.15.320.2.10 `exportAsDot()`** template<typename Settings>  
virtual void `smrat::cad::BaseProjection< Settings >`::`exportAsDot` (  
    std::ostream & ) const [inline], [virtual], [inherited]

Reimplemented in `smrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

**0.15.320.2.11 `freeID()`** template<typename Settings>  
void `smrat::cad::BaseProjection< Settings >`::`freeID` (  
    std::size\_t level,  
    std::size\_t id) [inline], [protected], [inherited]

Frees a currently used polynomial id for the given level.

**0.15.320.2.12 `getID()`** template<typename Settings>  
std::size\_t `smrat::cad::BaseProjection< Settings >`::`getID` (  
    std::size\_t level) [inline], [protected], [inherited]

Returns a fresh polynomial id for the given level.

**0.15.320.2.13 `getOrigin()`** template<typename Settings >  
 virtual `Origin smtrat::cad::BaseProjection< Settings >::getOrigin (`  
 `std::size_t level,`  
 `std::size_t id ) const [inline], [virtual], [inherited]`  
 Reimplemented in `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`.

**0.15.320.2.14 `getPolyForLifting()`** template<typename Settings >  
`OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (`  
 `std::size_t level,`  
 `SampleLiftedWith & slw ) [inline], [inherited]`  
 Get a polynomial from this level suited for lifting.

**0.15.320.2.15 `getPolynomialById()`** template<typename Settings >  
 virtual const `UPoly& smtrat::cad::BaseProjection< Settings >::getPolynomialById (`  
 `std::size_t level,`  
 `std::size_t id ) const [pure virtual], [inherited]`

Retrieves a polynomial from its id.

Implemented in `smtrat::qe::cad::Projection< Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::SIMPLE, BT, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

**0.15.320.2.16 `hasPolynomialById()`** template<typename Settings >  
 virtual bool `smtrat::cad::BaseProjection< Settings >::hasPolynomialById (`  
 `std::size_t level,`  
 `std::size_t id ) const [pure virtual], [inherited]`

Implemented in `smtrat::qe::cad::Projection< Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::SIMPLE, BT, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

**0.15.320.2.17 `isPurged()`** template<typename Settings >  
 bool `smtrat::cad::BaseProjection< Settings >::isPurged (`  
 `std::size_t level,`  
 `std::size_t id ) [inline], [protected], [inherited]`

**0.15.320.2.18 `removePolynomial()` [1/2]** template<typename Settings >  
 void `smtrat::cad::BaseProjection< Settings >::removePolynomial (`  
 `const Poly & p,`  
 `std::size_t cid,`  
 `bool isBound ) [inline], [inherited]`

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.320.2.19 `removePolynomial()` [2/2]** template<typename Settings >  
 virtual void `smtrat::cad::BaseProjection< Settings >::removePolynomial (`  
 `const UPoly & p,`  
 `std::size_t cid,`  
 `bool isBound ) [pure virtual], [inherited]`

Removes the given polynomial from the projection.

Implemented in `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::SIMPLE, BT, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::NONE, Backtracking::HIDE, Settings >`.

**0.15.320.2.20 `reset()`** template<typename Settings >  
void `smtrat::cad::BaseProjection< Settings >::reset ( )` [inline], [inherited]  
Resets all datastructures, use the given variables from now on.

**0.15.320.2.21 `setRemoveCallback()`** template<typename Settings >  
template<typename F >  
void `smtrat::cad::BaseProjection< Settings >::setRemoveCallback (`  
    `F && f )` [inline], [inherited]  
Sets a callback that is called whenever polynomials are removed.

**0.15.320.2.22 `size() [1/2]`** template<typename Settings >  
`std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const` [inline], [inherited]

**0.15.320.2.23 `size() [2/2]`** template<typename Settings >  
virtual `std::size_t smtrat::cad::BaseProjection< Settings >::size (`  
    `std::size_t level ) const` [pure virtual], [inherited]

Implemented in `smtrat::qe::cad::Projection< Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::INCREMENTAL, Backtracking::UNORDERED >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDDEN >`.

**0.15.320.2.24 `var()`** template<typename Settings >  
carl::Variable `smtrat::cad::BaseProjection< Settings >::var (`  
    `std::size_t level ) const` [inline], [protected], [inherited]  
Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.320.2.25 `vars()`** template<typename Settings >  
const auto& `smtrat::cad::BaseProjection< Settings >::vars ( ) const` [inline], [inherited]  
Returns the variables used for projection.

### 0.15.320.3 Field Documentation

**0.15.320.3.1 `mConstraints`** template<typename Settings >  
const `Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints` [protected], [inherited]

**0.15.320.3.2 `mInfo`** template<typename Settings >  
`ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo` [protected], [inherited]  
Additional info on projection, projection levels and projection polynomials.

**0.15.320.3.3 `mLiftingQueues`** template<typename Settings >  
`std::vector<PolynomialLiftingQueue<BaseProjection> > smtrat::cad::BaseProjection< Settings >::mLiftingQueues` [protected], [inherited]  
List of lifting queues that can be used for incremental projection.

**0.15.320.3.4 `mOperator`** template<typename Settings >  
`ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator` [protected], [inherited]  
The projection operator.

**0.15.320.3.5 mRemoveCallback** template<typename Settings >  
 std::function<void(std::size\_t, const SampleLiftedWith&) > smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]  
 Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

### 0.15.321 smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings > Class Template Reference

This class implements a projection that supports no incrementality and expects backtracking to be in order.  
`#include <Projection_Model.h>`

#### Public Member Functions

- `ModelBasedProjection` (const `Constraints` &c, const `Model` &model)
- `void reset()`  
*Resets all datastructures, use the given variables from now on.*
- `carl::Bitset addPolynomial` (const `UPoly` &p, std::size\_t cid, bool) override  
*Adds the given polynomial to the projection with the given constraint id as origin.*
- `void removePolynomial` (const `UPoly` &p, std::size\_t cid, bool) override  
*Removed the given polynomial from the projection.*
- `void projectNextLevel` (std::size\_t level)  
*Performs the projection model-based for one level, using only polynomials with closest roots.*
- `std::size_t size` (std::size\_t level) const override  
*Returns the number of polynomials in this level.*
- `bool empty` (std::size\_t level) const override  
*Returns whether the number of polynomials in this level is zero.*
- `carl::Bitset projectNewPolynomial` (const `ConstraintSelection` &=carl::Bitset(true))  
*Returns false, as the projection is not incremental.*
- `bool hasPolynomialById` (std::size\_t level, std::size\_t id) const override
- `const UPoly & getPolynomialById` (std::size\_t level, std::size\_t id) const override  
*Get the polynomial from this level with the given id.*
- `std::size_t dim()` const  
*Returns the dimension of the projection.*
- `virtual std::size_t size` (std::size\_t level) const=0
- `std::size_t size()` const
- `std::size_t dim()` const  
*Returns the dimension of the projection.*
- `const auto & vars()` const  
*Returns the variables used for projection.*
- `template<typename F> void setRemoveCallback(F &&f)`  
*Sets a callback that is called whenever polynomials are removed.*
- `carl::Bitset addPolynomial` (const `Poly` &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- `carl::Bitset addEqConstraint` (const `Poly` &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- `virtual carl::Bitset addEqConstraint` (const `UPoly` &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial of an equational constraint to the projection.*
- `void removePolynomial` (const `Poly` &p, std::size\_t cid, bool isBound)  
*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- `std::size_t size()` const

- virtual bool `empty ()`
- `OptionalID getPolyForLifting (std::size_t level, SampleLiftedWith &slw)`  
*Get a polynomial from this level suited for lifting.*
- virtual void `exportAsDot (std::ostream &) const`
- virtual `Origin getOrigin (std::size_t level, std::size_t id) const`

### Protected Member Functions

- void `callRemoveCallback (std::size_t level, const SampleLiftedWith &slw) const`
- `std::size_t getID (std::size_t level)`  
*Returns a fresh polynomial id for the given level.*
- void `freeID (std::size_t level, std::size_t id)`  
*Frees a currently used polynomial id for the given level.*
- `carl::Variable var (std::size_t level) const`  
*Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
- bool `canBePurgedByBounds (const UPoly &p) const`  
*Checks whether a polynomial can safely be ignored due to the bounds.*
- bool `isPurged (std::size_t level, std::size_t id)`

### Protected Attributes

- const `Constraints & mConstraints`
- `ProjectionInformation mlInfo`  
*Additional info on projection, projection levels and projection polynomials.*
- `std::function< void(std::size_t, const SampleLiftedWith &) > mRemoveCallback`  
*Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

### Friends

- template<typename S >  
`std::ostream & operator<< (std::ostream &os, const ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, S > &p)`

#### 0.15.321.1 Detailed Description

```
template<typename Settings>
class smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >
```

This class implements a projection that supports no incrementality and expects backtracking to be in order.  
It is based on the following data structures:

- `mPolynomialIDs`: maps polynomials to a (per level) unique id
- `mPolynomials`: stores polynomials as a list (per level) with their origin

The origin of a polynomial in level zero is the id of the corresponding constraint. For all other levels, it is the id of some polynomial from level zero such that the polynomial must be removed if the origin is removed. For a single projection operation, the resulting origin is the largest of the participating polynomials. If a polynomial is derived from multiple projection operations, the origin is the earliest and thus smallest, at least for this non-incremental setting.

#### 0.15.321.2 Constructor & Destructor Documentation

```
0.15.321.2.1 ModelBasedProjection() template<typename Settings >
smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::
::ModelBasedProjection (
 const Constraints & c,
 const Model & model) [inline]
```

### 0.15.321.3 Member Function Documentation

**0.15.321.3.1 addEqConstraint() [1/2]** template<typename Settings >
carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (
 const Poly & p,
 std::size\_t cid,
 bool isBound ) [inline], [inherited]

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.321.3.2 addEqConstraint() [2/2]** template<typename Settings >
virtual carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (
 const UPoly & p,
 std::size\_t cid,
 bool isBound ) [inline], [virtual], [inherited]

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in [smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >](#).

**0.15.321.3.3 addPolynomial() [1/2]** template<typename Settings >
carl::Bitset smrat::cad::BaseProjection< Settings >::addPolynomial (
 const Poly & p,
 std::size\_t cid,
 bool isBound ) [inline], [inherited]

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.321.3.4 addPolynomial() [2/2]** template<typename Settings >
carl::Bitset smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED,
Settings >::addPolynomial (
 const UPoly & p,
 std::size\_t cid,
 bool ) [inline], [override], [virtual]

Adds the given polynomial to the projection with the given constraint id as origin.

Asserts that the main variable of the polynomial is the first variable.

Implements [smrat::cad::BaseProjection< Settings >](#).

**0.15.321.3.5 callRemoveCallback()** template<typename Settings >
void smrat::cad::BaseProjection< Settings >::callRemoveCallback (
 std::size\_t level,
 const SampleLiftedWith & slw ) const [inline], [protected], [inherited]

**0.15.321.3.6 canBePurgedByBounds()** template<typename Settings >
bool smrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
 const UPoly & p ) const [inline], [protected], [inherited]

Checks whether a polynomial can safely be ignored due to the bounds.

**0.15.321.3.7 dim() [1/2]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::dim () const [inline], [inherited]  
Returns the dimension of the projection.

**0.15.321.3.8 dim() [2/2]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::dim [inline]  
Returns the dimension of the projection.

**0.15.321.3.9 empty() [1/2]** template<typename Settings >  
virtual bool smtrat::cad::BaseProjection< Settings >::empty () [inline], [virtual], [inherited]

**0.15.321.3.10 empty() [2/2]** template<typename Settings >  
bool smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::empty (  
 std::size\_t level ) const [inline], [override], [virtual]

Returns whether the number of polynomials in this level is zero.

Implements smtrat::cad::BaseProjection< Settings >.

**0.15.321.3.11 exportAsDot()** template<typename Settings >  
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (

std::ostream & ) const [inline], [virtual], [inherited]

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

**0.15.321.3.12 freeID()** template<typename Settings >  
void smtrat::cad::BaseProjection< Settings >::freeID (

std::size\_t level,

std::size\_t id ) [inline], [protected], [inherited]

Frees a currently used polynomial id for the given level.

**0.15.321.3.13 getID()** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::getID (

std::size\_t level ) [inline], [protected], [inherited]

Returns a fresh polynomial id for the given level.

**0.15.321.3.14 getOrigin()** template<typename Settings >  
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (

std::size\_t level,

std::size\_t id ) const [inline], [virtual], [inherited]

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

**0.15.321.3.15 getPolyForLifting()** template<typename Settings >  
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (

std::size\_t level,

SampleLiftedWith & slw ) [inline], [inherited]

Get a polynomial from this level suited for lifting.

**0.15.321.3.16 `getPolynomialById()`** template<typename Settings >  
 const UPoly& smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::getPolynomialById ( std::size\_t level, std::size\_t id ) const [inline], [override], [virtual]  
 Get the polynomial from this level with the given id.  
 Implements smtrat::cad::BaseProjection< Settings >.

**0.15.321.3.17 `hasPolynomialById()`** template<typename Settings >  
 bool smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::hasPolynomialById ( std::size\_t level, std::size\_t id ) const [inline], [override], [virtual]  
 Implements smtrat::cad::BaseProjection< Settings >.

**0.15.321.3.18 `isPurged()`** template<typename Settings >  
 bool smtrat::cad::BaseProjection< Settings >::isPurged ( std::size\_t level, std::size\_t id ) [inline], [protected], [inherited]

**0.15.321.3.19 `projectNewPolynomial()`** template<typename Settings >  
 carl::Bitset smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::projectNewPolynomial ( const ConstraintSelection & = carl::Bitset(true) ) [inline]  
 Returns false, as the projection is not incremental.

**0.15.321.3.20 `projectNextLevel()`** template<typename Settings >  
 void smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::projectNextLevel ( std::size\_t level ) [inline]  
 Performs the projection model-based for one level, using only polynomials with closest roots.

**0.15.321.3.21 `removePolynomial()` [1/2]** template<typename Settings >  
 void smtrat::cad::BaseProjection< Settings >::removePolynomial ( const Poly & p, std::size\_t cid, bool isBound ) [inline], [inherited]

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.321.3.22 `removePolynomial()` [2/2]** template<typename Settings >  
 void smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::removePolynomial ( const UPoly & p, std::size\_t cid, bool ) [inline], [override], [virtual]

Removed the given polynomial from the projection.

Asserts that this polynomial was the one added last and has the given constraint id as origin. Calls the callback function for every level with a mask designating the polynomials removed from this level.

Implements smtrat::cad::BaseProjection< Settings >.

**0.15.321.3.23 `reset()`** template<typename Settings >  
void smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::reset () [inline]  
Resets all datastructures, use the given variables from now on.

**0.15.321.3.24 `setRemoveCallback()`** template<typename Settings >  
template<typename F >  
void smrat::cad::BaseProjection< Settings >::setRemoveCallback (  
 F && f ) [inline], [inherited]  
Sets a callback that is called whenever polynomials are removed.

**0.15.321.3.25 `size() [1/4]`** template<typename Settings >  
std::size\_t smrat::cad::BaseProjection< Settings >::size () const [inline], [inherited]

**0.15.321.3.26 `size() [2/4]`** template<typename Settings >  
std::size\_t smrat::cad::BaseProjection< Settings >::size (  
 void ) [inline]

**0.15.321.3.27 `size() [3/4]`** template<typename Settings >  
std::size\_t smrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, Settings >::size (  
 std::size\_t level ) const [inline], [override], [virtual]  
Returns the number of polynomials in this level.  
Implements `smrat::cad::BaseProjection< Settings >`.

**0.15.321.3.28 `size() [4/4]`** template<typename Settings >  
virtual std::size\_t smrat::cad::BaseProjection< Settings >::size (  
 void )

**0.15.321.3.29 `var()`** template<typename Settings >  
carl::Variable smrat::cad::BaseProjection< Settings >::var (  
 std::size\_t level ) const [inline], [protected], [inherited]  
Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.321.3.30 `vars()`** template<typename Settings >  
const auto& smrat::cad::BaseProjection< Settings >::vars () const [inline], [inherited]  
Returns the variables used for projection.

## 0.15.321.4 Friends And Related Function Documentation

**0.15.321.4.1 `operator<<`** template<typename Settings >  
template<typename S >  
std::ostream& operator<< (  
 std::ostream & os,  
 const ModelBasedProjection< Incrementality::NONE, Backtracking::ORDERED, S > & p  
) [friend]

### 0.15.321.5 Field Documentation

**0.15.321.5.1 mConstraints** template<typename Settings >  
 const [Constraints](#)& [smtrat::cad::BaseProjection](#)< Settings >::mConstraints [protected], [inherited]

**0.15.321.5.2 mInfo** template<typename Settings >  
[ProjectionInformation](#) [smtrat::cad::BaseProjection](#)< Settings >::mInfo [protected], [inherited]  
 Additional info on projection, projection levels and projection polynomials.

**0.15.321.5.3 mRemoveCallback** template<typename Settings >  
 std::function<void(std::size\_t, const [SampleLiftedWith](#)&)> [smtrat::cad::BaseProjection](#)< Settings >::mRemoveCallback [protected], [inherited]  
 Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

## 0.15.322 smtrat::Module Class Reference

A base class for all kind of theory solving methods.

```
#include <Module.h>
```

### Data Structures

- struct [Lemma](#)
- struct [ModuleStatistics](#)
- struct [TheoryPropagation](#)

### Public Types

- enum class [LemmaType](#) : unsigned { [NORMAL](#) = 0 , [PERMANENT](#) = 1 }

### Public Member Functions

- **Module** (const [ModuleInput](#) \*\_formula, [Conditionals](#) &\_foundAnswer, [Manager](#) \*\_manager=nullptr, std::string module\_name="Module")  
*Constructs a module.*
- virtual ~[Module](#) ()  
*Destructs a module.*
- bool [inform](#) (const [FormulaT](#) &\_constraint)  
*Informs the module about the given constraint.*
- void [deinform](#) (const [FormulaT](#) &\_constraint)  
*The inverse of informing about a constraint.*
- virtual void [init](#) ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- bool [add](#) ([ModuleInput](#)::const\_iterator \_subformula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual [Answer check](#) (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_← VARIABLE)  
*Checks the received formula for consistency.*
- virtual void [remove](#) ([ModuleInput](#)::const\_iterator \_subformula)  
*Removes everything related to the given sub-formula of the received formula.*
- virtual void [updateModel](#) () const

*Updates the model, if the solver has detected the consistency of the received formula, beforehand.*

- virtual void `updateAllModels ()`

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*

- virtual std::list< std::vector< carl::Variable > > `getModelEqualities () const`

*Partition the variables from the current model into equivalence classes according to their assigned value.*

- unsigned `currentlySatisfiedByBackend (const FormulaT &_formula) const`
- virtual unsigned `currentlySatisfied (const FormulaT &) const`
- bool `receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- std::size\_t `id () const`
- void `setId (std::size_t _id)`

*Sets this module's unique ID to identify itself.*

- `thread_priority threadPriority () const`
- void `setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*

- const `ModuleInput * pReceivedFormula () const`
- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const std::vector< Model > & `allModels () const`
- const std::vector< FormulaSetT > & `infeasibleSubsets () const`
- const std::vector< Module \* > & `usedBackends () const`
- const carl::FastSet< FormulaT > & `constraintsToInform () const`
- const carl::FastSet< FormulaT > & `informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*

- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*

- void `clearLemmas ()`

*Deletes all yet found lemmas.*

- const std::vector< Lemma > & `lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*

- const `smrat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- virtual std::string `moduleName () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*

- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*

- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*

- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `isValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`

- Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, FormulaT > **getReceivedFormulaSimplified** ()
  - void **print** (const std::string &\_initiation="\*\*\*") const  
*Prints everything relevant of the solver.*
  - void **printReceivedFormula** (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of the received formula.*
  - void **printPassedFormula** (const std::string &\_initiation="\*\*\*") const  
*Prints the vector of passed formula.*
  - void **printInfeasibleSubsets** (const std::string &\_initiation="\*\*\*") const  
*Prints the infeasible subsets.*
  - void **printModel** (std::ostream &\_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
  - void **printAllModels** (std::ostream &\_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void **freeSplittingVariable** (const FormulaT &\_splittingVariable)

### Static Public Attributes

- static size\_t **mNumOfBranchVarsToStore** = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > **mLastBranches** = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t **mFirstPosInLastBranches** = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > **mOldSplittingVariables**  
*Reusable splitting variables.*

### Protected Member Functions

- virtual bool **informCore** (const FormulaT &\_constraint)  
*Informs the module about the given constraint.*
- virtual void **deinformCore** (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- virtual bool **addCore** (ModuleInput::const\_iterator formula)  
*The module has to take the given sub-formula of the received formula into account.*
- virtual **Answer checkCore** ()  
*Checks the received formula for consistency.*
- virtual void **removeCore** (ModuleInput::const\_iterator formula)  
*Removes everything related to the given sub-formula of the received formula.*
- bool **anAnswerFound** () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void **clearModel** () const  
*Clears the assignment, if any was found.*
- void **clearModels** () const  
*Clears all assignments, if any was found.*
- void **cleanModel** () const

- Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin ()`
  - `ModuleInput::iterator passedFormulaEnd ()`
  - `void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
  - `const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`

*Gets the origins of the passed formula at the given position.*
  - `std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
  - `std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
  - `void informBackends (const FormulaT &_constraint)`

*Informs all backends of this module about the given constraint.*
  - `virtual void addConstraintToInform (const FormulaT &_constraint)`

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
  - `std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`

*Copies the given sub-formula of the received formula to the passed formula.*
  - `bool originInReceivedFormula (const FormulaT &_origin) const`
  - `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)`

*Adds the given formula to the passed formula with no origin.*
  - `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`

*Adds the given formula to the passed formula.*
  - `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`

*Adds the given formula to the passed formula.*
  - `void generateTrivialInfeasibleSubset ()`

*Stores the trivial infeasible subset being the set of received formulas.*
  - `void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

*Stores an infeasible subset consisting only of the given received formula.*
  - `std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const`
  - `void getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
  - `void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
  - `void getInfeasibleSubsets ()`

*Copies the infeasible subsets of the passed formula.*
  - `const Model & backendsModel () const`

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - `void getBackendsModel () const`
  - `void getBackendsAllModels () const`

*Stores all models of a backend in the list of all models of this module.*
  - `virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*
  - `virtual Answer runBackends ()`
  - `virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
  - `void clearPassedFormula ()`
  - `std::vector< FormulaSetT > getInfeasibleSubsets (const Module &backend) const`

*Get the infeasible subsets the given backend provides.*
  - `std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`

*Merges the two vectors of sets into the first one.*

- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename `Poly::PolyType` &`_branchingPolynomial`, const `Rational` &`_branchingValue`) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const `Poly` &`_polynomial`, bool `_integral`, const `Rational` &`_value`, std::vector< `FormulaT` > &&`_premise`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (const `Poly` &`_polynomial`, bool `_integral`, const `Rational` &`_value`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`, const std::vector< `FormulaT` > &`_premise=std::vector< FormulaT >()`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (carl::Variable::Arg `_var`, const `Rational` &`_value`, std::vector< `FormulaT` > &&`_premise`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- bool `branchAt` (carl::Variable::Arg `_var`, const `Rational` &`_value`, bool `_leftCaseWeak=true`, bool `_preferLeftCase=true`, bool `_useReceivedFormulaAsPremise=false`, const std::vector< `FormulaT` > &`_premise=std::vector< FormulaT >()`)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.*
- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint `p!=0`.*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &`formula`)
 

*Adds a formula to the `InformationRelevantFormula`.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
 

*Gets all `InformationRelevantFormulas`.*
- bool `isLemmaLevel` (`LemmaLevel` `level`)
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &`_modelA`, const `Model` &`_modelB`)
 

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- `std::vector< Module * > mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- `std::vector< Module * > mAllBackends`  
*The backends of this module which have been used.*
- `std::vector< Lemma > mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- `carl::FastSet< FormulaT > mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.322.1 Detailed Description

A base class for all kind of theory solving methods.

### 0.15.322.2 Member Enumeration Documentation

#### 0.15.322.2.1 LemmaType `enum smtrat::Module::LemmaType : unsigned [strong]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.322.3 Constructor & Destructor Documentation

#### 0.15.322.3.1 Module() `smtrat::Module::Module (`

```
 const ModuleInput * _formula,
 Conditionals & _foundAnswer,
 Manager * _manager = nullptr,
 std::string module_name = "Module")
```

Constructs a module.

#### Parameters

|                       |                                                             |
|-----------------------|-------------------------------------------------------------|
| <code>_formula</code> | The formula passed to this module, called received formula. |
|-----------------------|-------------------------------------------------------------|

**Parameters**

|                           |                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------|
| <code>_foundAnswer</code> | Vector of Booleans: If any of them is true, we have to terminate a running check procedure. |
| <code>_manager</code>     | A reference to the manager of the solver using this module.                                 |
| <code>module_name</code>  | Name of this module as string for printing.                                                 |

**0.15.322.3.2 ~Module()** `smtrat::Module::~Module () [virtual]`  
Destructs a module.

**0.15.322.4 Member Function Documentation**

**0.15.322.4.1 add()** `bool smtrat::Module::add (ModuleInput::const_iterator _subformula )`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.322.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (const FormulaT & _constraint ) [protected], [virtual]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.322.4.3 addCore()** `virtual bool smtrat::Module::addCore (ModuleInput::const_iterator formula ) [inline], [protected], [virtual]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#),

`smrat::UnionFindModule< Settings >, smrat::UFCegarModule< Settings >, smrat::STropModule< Settings >, smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntEqModule< Settings >, smrat::IntBlastModule< Settings >, smrat::IncWidthModule< Settings >, smrat::ICEModule< Settings >, smrat::FPPModule< Settings >, smrat::FouMoModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::CubeLIAModule< Settings >, smrat::CSplitModule< Settings >, smrat::BVMModule< Settings >, and smrat::GBModule< Settings >.`

#### 0.15.322.4.4 `addInformationRelevantFormula()` `void smrat::Module::addInformationRelevantFormula(`

`const FormulaT & formula ) [protected]`

Adds a formula to the InformationRelevantFormula.

##### Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

#### 0.15.322.4.5 `addLemma()` `void smrat::Module::addLemma(`

`const FormulaT & _lemma,`

`const LemmaType & _lt = LemmaType::NORMAL,`

`const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline]`

Stores a lemma being a valid formula.

##### Parameters

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

#### 0.15.322.4.6 `addOrigin()` `void smrat::Module::addOrigin(`

`ModuleInput::iterator _formula,`

`const FormulaT & _origin ) [inline], [protected]`

Adds the given set of formulas in the received formula to the origins of the given passed formula.

##### Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

#### 0.15.322.4.7 `addReceivedSubformulaToPassedFormula()` `std::pair<ModuleInput::iterator,bool>`

`smrat::Module::addReceivedSubformulaToPassedFormula(`

`ModuleInput::const_iterator _subformula ) [inline], [protected]`

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

##### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

```
0.15.322.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat→
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula. |
|-----------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.322.4.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat→
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected]
```

Adds the given formula to the passed formula.

#### Parameters

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                 |
| <i>_origin</i>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.322.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected]
```

Adds the given formula to the passed formula.

#### Parameters

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.322.4.11 allModels()** const std::vector<[Model](#)>& smtrat::Module::allModels () const [inline]**Returns**

All satisfying assignments, if existent.

**0.15.322.4.12 anAnswerFound()** bool smtrat::Module::anAnswerFound () const [inline], [protected]  
Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.**Returns**

True, if one of them has determined a result.

**0.15.322.4.13 answerFound()** const [smtusat::Conditionals](#)& smtrat::Module::answerFound () const [inline]**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.322.4.14 backendsModel()** const [Model](#) & smtrat::Module::backendsModel () const [protected]  
Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.**0.15.322.4.15 branchAt() [1/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const [Rational](#) & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< [FormulaT](#) > & \_premise = std::vector<[FormulaT](#)>() ) [inline], [protected]**0.15.322.4.16 branchAt() [2/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const [Rational](#) & \_value, std::vector< [FormulaT](#) > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected]

```
0.15.322.4.17 branchAt() [3/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector< FormulaT >()) [inline],
[protected]
```

```
0.15.322.4.18 branchAt() [4/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.322.4.19 check() Answer smtrat::Module::check (
```

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::PModule](#).

**0.15.322.4.20 checkCore()** [Answer](#) `smrat::Module::checkCore () [protected], [virtual]`

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::SymmetryModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::SplitSOSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::MCBModule< Settings >](#), [smrat::LVEModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::ICEModule< Settings >](#), [smrat::GBPPModule< Settings >](#), [smrat::GBModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::ESModule< Settings >](#), [smrat::EQPreprocessingModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::EMModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::CNFerModule](#), [smrat::BVMModule< Settings >](#), and [smrat::BEModule< Settings >](#).

**0.15.322.4.21 checkInfSubsetForMinimality()** `void smrat::Module::checkInfSubsetForMinimality (`

```
 std::vector< FormulaSetT >::const_iterator _infsSubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.322.4.22 checkModel()** `unsigned smrat::Module::checkModel () const [protected]`**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.322.4.23 cleanModel()** `void smrat::Module::cleanModel () const [inline], [protected]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.322.4.24 `clearLemmas()`** `void smtrat::Module::clearLemmas () [inline]`  
 Deletes all yet found lemmas.

**0.15.322.4.25 `clearModel()`** `void smtrat::Module::clearModel () const [inline], [protected]`  
 Clears the assignment, if any was found.

**0.15.322.4.26 `clearModels()`** `void smtrat::Module::clearModels () const [inline], [protected]`  
 Clears all assignments, if any was found.

**0.15.322.4.27 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula () [protected]`

**0.15.322.4.28 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins (`  
`const FormulaT & _formula,`  
`FormulaSetT & _origins ) const`

**0.15.322.4.29 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins (`  
`const FormulaT & _formula,`  
`FormulasT & _origins ) const`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.322.4.30 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations ()`

**0.15.322.4.31 `constraintsToInform()`** `const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline]`

#### Returns

The constraints which the backends must be informed about.

**0.15.322.4.32 `currentlySatisfied()`** `virtual unsigned smtrat::Module::currentlySatisfied (`  
`const FormulaT & ) const [inline], [virtual]`

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, and `smtrat::LRAModule< LRASettings >`

**0.15.322.4.33 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.322.4.34 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint )`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.322.4.35 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASetting >](#)

**0.15.322.4.36 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.322.4.37 eraseSubformulaFromPassedFormula()** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to remove from the passed formula. |
|--------------------------|----------------------------------------------------|

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|

**Returns**

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.322.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel() const`  
Excludes all variables from the current model, which do not occur in the received formula.

**0.15.322.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin(`  
`const std::vector< FormulaT > & _origins ) const [protected]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.322.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass() const [inline]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.322.4.41 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula() const [inline]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.322.4.42 `freeSplittingVariable()`** `static void smtrat::Module::freeSplittingVariable(`  
`const FormulaT & _splittingVariable ) [inline], [static]`

**0.15.322.4.43 `generateTrivialInfeasibleSubset()`** `void smtrat::Module::generateTrivialInfeasibleSubset() [inline], [protected]`  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.322.4.44 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels () const [protected]  
Stores all models of a backend in the list of all models of this module.

**0.15.322.4.45 getBackendsModel()** void smtrat::Module::getBackendsModel () const [protected]

**0.15.322.4.46 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets () const [protected]  
Copies the infeasible subsets of the passed formula.

**0.15.322.4.47 getInfeasibleSubsets() [2/2]** std::vector< [FormulaSetT](#) > smtrat::Module::getInfeasibleSubsets (const [Module](#) & [\\_backend](#)) const [protected]  
Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <a href="#">_backend</a> | The backend from which to obtain the infeasible subsets. |
|--------------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.322.4.48 getInformationRelevantFormulas()** const std::set< [FormulaT](#) > & smtrat::Module::getInformationRelevantFormulas () const [protected]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.322.4.49 getModelEqualities()** std::list< std::vector< [carl::Variable](#) > > smtrat::Module::getModelEqualities () const [virtual]  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.322.4.50 getOrigins() [1/3]** void smtrat::Module::getOrigins (const [FormulaT](#) & [\\_formula](#), [FormulaSetT](#) & [\\_origins](#)) const [inline], [protected]

#### Parameters

|                          |  |
|--------------------------|--|
| <a href="#">_formula</a> |  |
| <a href="#">_origins</a> |  |

```
0.15.322.4.51 getOrigins() [2/3] void smtrat::Module::getOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inline], [protected]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

```
0.15.322.4.52 getOrigins() [3/3] const FormulaT& smtrat::Module::getOrigins (
 ModuleInput::const_iterator _formula) const [inline], [protected]
```

Gets the origins of the passed formula at the given position.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

```
0.15.322.4.53 getReceivedFormulaSimplified() std::pair< bool, FormulaT > smtrat::Module::get<-
ReceivedFormulaSimplified () [virtual]
```

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

```
0.15.322.4.54 hasLemmas() bool smtrat::Module::hasLemmas () [inline]
```

Checks whether this module or any of its backends provides any lemmas.

```
0.15.322.4.55 hasValidInfeasibleSubset() bool smtrat::Module::hasValidInfeasibleSubset () const
```

#### Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

```
0.15.322.4.56 id() std::size_t smtrat::Module::id () const [inline]
```

#### Returns

A unique ID to identify this module instance.

**0.15.322.4.57 infeasibleSubsets()** const std::vector<[FormulaSetT](#)>& smtrat::Module::infeasibleSubsets ( ) const [inline]

#### Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.322.4.58 inform()** bool smtrat::Module::inform ( const [FormulaT](#) & \_constraint )

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.322.4.59 informBackends()** void smtrat::Module::informBackends ( const [FormulaT](#) & \_constraint ) [inline], [protected]

Informs all backends of this module about the given constraint.

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.322.4.60 informCore()** virtual bool smtrat::Module::informCore ( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::BVMModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.322.4.61 informedConstraints()** const carl::FastSet<[FormulaT](#)>& smtrat::Module::informedConstraints ( ) const [inline]

#### Returns

The position of the first constraint of which no backend has been informed about.

**0.15.322.4.62 init()** void smtrat::Module::init ( ) [virtual]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), and [smtrat::BVModule< Settings >](#).

**0.15.322.4.63 is\_minimizing()** bool smtrat::Module::is\_minimizing ( ) const [inline]

**0.15.322.4.64 isLemmaLevel()** bool smtrat::Module::isLemmaLevel (

[LemmaLevel](#) level ) [protected]

Checks if current lemma level is greater or equal to given level.

#### Parameters

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.322.4.65 isPreprocessor()** bool smtrat::Module::isPreprocessor ( ) const [inline]

#### Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.322.4.66 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline]

#### Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.322.4.67 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge (

const std::vector< [FormulaT](#) > & \_vecSetA,

const std::vector< [FormulaT](#) > & \_vecSetB ) const [protected]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

#### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.322.4.68 model()** const `Model&` `smtrat::Module::model()` const [inline]**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.322.4.69 modelsDisjoint()** bool `smtrat::Module::modelsDisjoint()`

```
const Model & _modelA,
 const Model & _modelB) [static], [protected]
```

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.322.4.70 moduleName()** virtual std::string `smtrat::Module::moduleName()` const [inline], [virtual]**Returns**

The name of the given module type as name.

Reimplemented in `smtrat::VSModule< Settings >`, `smtrat::UnionFindModule< Settings >`, `smtrat::UFCegarModule< Settings >`, `smtrat::STropModule< Settings >`, `smtrat::SplitSOSModule< Settings >`, `smtrat::PBPPModule< Settings >`, `smtrat::PBGaussModule< Settings >`, `smtrat::NRAILModule< Settings >`, `smtrat::NewCADModule< Settings >`, `smtrat::LVEModule< Settings >`, `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, `smtrat::LRAModule< LRASettings1 >`, `smtrat::IntEqModule< Settings >`, `smtrat::IntBlastModule< Settings >`, `smtrat::IncWidthModule< Settings >`, `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, `smtrat::ICPModule< ICPSettings1 >`, `smtrat::GBModule< Settings >`, `smtrat::FouMoModule< Settings >`, `smtrat::EQPreprocessingModule< Settings >`, `smtrat::EQModule< Settings >`, `smtrat::CurryModule< Settings >`, `smtrat::CubeLIAModule< Settings >`, `smtrat::CSplitModule< Settings >`, `smtrat::BVModule< Settings >`, and `smtrat::BEModule< Settings >`.

**0.15.322.4.71 objective()** carl::Variable `smtrat::Module::objective()` const [inline]**0.15.322.4.72 originInReceivedFormula()** bool `smtrat::Module::originInReceivedFormula()`

```
const FormulaT & _origin) const [protected]
```

**0.15.322.4.73 passedFormulaBegin()** `ModuleInput::iterator` `smtrat::Module::passedFormulaBegin()` [inline], [protected]**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.322.4.74 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ()`  
[inline], [protected]

**Returns**

An iterator to the end of the passed formula.

**0.15.322.4.75 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula () const`  
[inline]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.322.4.76 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula () const`  
[inline]

**Returns**

A pointer to the formula passed to this module.

**0.15.322.4.77 print()** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const`  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.322.4.78 printAllModels()** `void smtrat::Module::printAllModels (`  
`std::ostream & _out = std::cout )`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.322.4.79 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets (`  
`const std::string & _initiation = "***" ) const`

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.322.4.80 printModel()** `void smtrat::Module::printModel (`  
`std::ostream & _out = std::cout ) const`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.322.4.81 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const`

Prints the vector of passed formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.322.4.82 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const`

Prints the vector of the received formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.322.4.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.322.4.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline]`

Notifies that the received formulas has been checked.

**0.15.322.4.85 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.322.4.86 receivedVariable()** `bool smrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline]`

**0.15.322.4.87 remove()** `void smrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

**0.15.322.4.88 removeCore()** `virtual void smrat::Module::removeCore ( ModuleInput::const_iterator formula ) [inline], [protected], [virtual]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.322.4.89 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected]`

**0.15.322.4.90 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected]`

**0.15.322.4.91 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const [inline]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.322.4.92 rReceivedFormula()** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline]`

**Returns**

A reference to the formula passed to this module.

**0.15.322.4.93 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual]`  
Reimplemented in [smtrat::PModule](#).

**0.15.322.4.94 runBackends() [2/2]** `Answer smtrat::Module::runBackends ( bool _final, bool _full, carl::Variable _objective ) [protected], [virtual]`

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.322.4.95 setId()** `void smtrat::Module::setId ( std::size_t _id ) [inline]`

Sets this modules unique ID to identify itself.

**Parameters**

|                                                                       |                                    |
|-----------------------------------------------------------------------|------------------------------------|
| $\leftrightarrow$<br>$\underline{\leftrightarrow}$<br><code>id</code> | The id to set this module's id to. |
|-----------------------------------------------------------------------|------------------------------------|

**0.15.322.4.96 setThreadPriority()** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline]`

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.322.4.97 solverState()** `Answer smtrat::Module::solverState ( ) const [inline]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.322.4.98 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.322.4.99 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.322.4.100 updateAllModels()** `void smtrat::Module::updateAllModels ( ) [virtual]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.322.4.101 updateLemmas()** `void smtrat::Module::updateLemmas ( )`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.322.4.102 updateModel()** `void smtrat::Module::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::PModule](#), [smtrat::VSMModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::MCBModule< Settings >](#), [smtrat::LVEModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::GBPPModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::ESModule< Settings >](#), [smtrat::EQPreprocessingModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), and [smtrat::BVMModule< Settings >](#).

**0.15.322.4.103 usedBackends()** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.322.5 Field Documentation

**0.15.322.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends` [protected]  
The backends of this module which have been used.

**0.15.322.5.2 mAllModels** `std::vector<Model> smtrat::Module::mAllModels` [mutable], [protected]  
Stores all satisfying assignments.

**0.15.322.5.3 mBackendsFoundAnswer** `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer` [protected]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.322.5.4 mConstraintsToInform** `carl::FastSet<FormulaT> smtrat::Module::mConstraintsToInform` [protected]  
Stores the constraints which the backends must be informed about.

**0.15.322.5.5 mFinalCheck** `bool smtrat::Module::mFinalCheck` [protected]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.322.5.6 mFirstPosInLastBranches** `std::size_t smtrat::Module::mFirstPosInLastBranches = 0` [static]  
The beginning of the cyclic buffer storing the last branches.

**0.15.322.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smtrat::Module::mFirstSubformulaToPass` [protected]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.322.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula` [protected]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.322.5.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer` [protected]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.322.5.10 mFullCheck** `bool smtrat::Module::mFullCheck` [protected]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.322.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets` [protected]  
Stores the infeasible subsets.

**0.15.322.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints` [protected]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.322.5.13 mLastBranches** `std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static]  
Stores the last branches in a cycle buffer.

**0.15.322.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas` [protected]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.322.5.15 mModel** `Model smtrat::Module::mModel` [mutable], [protected]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.322.5.16 mModelComputed** `bool smtrat::Module::mModelComputed` [mutable], [protected]  
True, if the model has already been computed.

**0.15.322.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5` [static]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.322.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable` [protected]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.322.5.19 mOldSplittingVariables** `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables` [static]  
Reusable splitting variables.

**0.15.322.5.20 mpManager** `Manager* const smtrat::Module::mpManager` [protected]  
A reference to the manager.

**0.15.322.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter` [mutable], [protected]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.322.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.322.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations` [protected]

**0.15.322.5.24 mUsedBackends** std::vector<[Module](#)\*> smrat::Module::mUsedBackends [protected]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.322.5.25 mVariableCounters** std::vector<std::size\_t> smrat::Module::mVariableCounters [protected]  
Maps variables to the number of their occurrences.

## 0.15.323 smrat::ModuleFactory< Module > Struct Template Reference

```
#include <StrategyGraph.h>
```

### Public Member Functions

- [ModuleFactory \(\)](#)
- [Module \\* create \(const ModuleInput \\*\\_formula, Conditionals &\\_conditionals, Manager \\*const \\_manager\)](#)
- [std::string moduleName \(\) const](#)

### 0.15.323.1 Constructor & Destructor Documentation

**0.15.323.1.1 ModuleFactory()** template<typename Module >  
[smrat::ModuleFactory< Module >::ModuleFactory \(\)](#) [inline]

### 0.15.323.2 Member Function Documentation

**0.15.323.2.1 create()** template<typename Module >  
[Module\\* smrat::ModuleFactory< Module >::create \(const ModuleInput \\* \\_formula, Conditionals & \\_conditionals, Manager \\*const \\_manager \)](#) [inline], [virtual]  
Implements [smrat::AbstractModuleFactory](#).

**0.15.323.2.2 moduleName()** template<typename Module >  
[std::string smrat::ModuleFactory< Module >::moduleName \(\) const](#) [inline], [virtual]  
Implements [smrat::AbstractModuleFactory](#).

## 0.15.324 smrat::ModuleInput Class Reference

The input formula a module has to consider for it's satisfiability check.

```
#include <ModuleInput.h>
```

### Data Structures

- struct [IteratorCompare](#)

### Public Types

- typedef super::iterator [iterator](#)  
*Passing through the list::iterator.*
- typedef super::const\_iterator [const\\_iterator](#)  
*Passing through the list::const\_iterator.*

## Public Member Functions

- `ModuleInput ()`  
`Constructs a module input.`
- `carl::Condition properties () const`
- `bool is_constraint_conjunction () const`
- `bool isConstraintLiteralConjunction () const`
- `bool is_real_constraint_conjunction () const`
- `bool isRealConstraintLiteralConjunction () const`
- `bool is_integer_constraint_conjunction () const`
- `bool isIntegerConstraintLiteralConjunction () const`
- `bool containsBitVectorConstraints () const`
- `bool containsBooleanVariables () const`
- `bool containsRealVariables () const`
- `bool containsIntegerVariables () const`
- `bool containsUninterpretedEquations () const`
- `bool is_only_propositional () const`
- `unsigned satisfiedBy (const Model &_assignment) const`
- `unsigned satisfiedBy (const RationalAssignment &_assignment) const`
- `const auto & back () const`
- `iterator begin ()`
- `iterator end ()`
- `const_iterator begin () const`
- `const_iterator end () const`
- `auto rbegin () const`
- `auto rend () const`
- `bool empty () const`
- `size_t size () const`
- `iterator find (const FormulaT &_formula)`
- `const_iterator find (const FormulaT &_formula) const`
- `iterator find (const_iterator _hint, const FormulaT &_formula)`
- `const_iterator find (const_iterator _hint, const FormulaT &_formula) const`
- `bool contains (const FormulaT &_subformula) const`
- `void updateProperties ()`  
`Updates all properties of the formula underlying this module input.`
- `void gatherVariables (carl::carlVariables &vars) const`
- `operator FormulaT () const`
- `void addOrigin (iterator _formula, const FormulaT &_origin)`
- `iterator erase (iterator _formula)`
- `void clearOrigins (iterator _formula)`
- `bool removeOrigin (iterator _formula, const FormulaT &_origin)`
- `bool removeOrigins (iterator _formula, const std::shared_ptr< FormulasT > &_origins)`
- `std::pair< iterator, bool > add (const FormulaT &_formula, bool _mightBeConjunction=true)`
- `std::pair< iterator, bool > add (const FormulaT &_formula, const FormulaT &_origins, bool _mightBeConjunction=true)`
- `std::pair< iterator, bool > add (const FormulaT &_formula, const std::shared_ptr< FormulasT > &_origins, bool _mightBeConjunction=true)`
- `std::pair< iterator, bool > add (const FormulaT &_formula, bool _hasSingleOrigin, const FormulaT &_origin, const std::shared_ptr< FormulasT > &_origins, bool _mightBeConjunction=true)`

## Friends

- class `Module`
- class `Manager`

### 0.15.324.1 Detailed Description

The input formula a module has to consider for it's satisfiability check.  
It is a list of formulas and semantically considered as their conjunction.

### 0.15.324.2 Member Typedef Documentation

#### 0.15.324.2.1 `const_iterator` `typedef super::const_iterator smtrat::ModuleInput::const_iterator`

Passing through the list::const\_iterator.

#### 0.15.324.2.2 `iterator` `typedef super::iterator smtrat::ModuleInput::iterator`

Passing through the list::iterator.

### 0.15.324.3 Constructor & Destructor Documentation

#### 0.15.324.3.1 `ModuleInput()` `smtrat::ModuleInput::ModuleInput ( ) [inline]`

Constructs a module input.

### 0.15.324.4 Member Function Documentation

#### 0.15.324.4.1 `add() [1/4]` `std::pair<iterator,bool> smtrat::ModuleInput::add ( const FormulaT & _formula, bool _hasSingleOrigin, const FormulaT & _origin, const std::shared_ptr< FormulasT > & _origins, bool _mightBeConjunction = true )`

#### 0.15.324.4.2 `add() [2/4]` `std::pair<iterator,bool> smtrat::ModuleInput::add ( const FormulaT & _formula, bool _mightBeConjunction = true ) [inline]`

#### 0.15.324.4.3 `add() [3/4]` `std::pair<iterator,bool> smtrat::ModuleInput::add ( const FormulaT & _formula, const FormulaT & _origins, bool _mightBeConjunction = true ) [inline]`

#### 0.15.324.4.4 `add() [4/4]` `std::pair<iterator,bool> smtrat::ModuleInput::add ( const FormulaT & _formula, const std::shared_ptr< FormulasT > & _origins, bool _mightBeConjunction = true ) [inline]`

#### 0.15.324.4.5 `addOrigin()` `void smtrat::ModuleInput::addOrigin ( iterator _formula, const FormulaT & _origin ) [inline]`

**0.15.324.4.6 `back()`** `const auto& smtrat::ModuleInput::back ( ) const [inline]`

**0.15.324.4.7 `begin() [1/2]`** `iterator smtrat::ModuleInput::begin ( ) [inline]`

**0.15.324.4.8 `begin() [2/2]`** `const_iterator smtrat::ModuleInput::begin ( ) const [inline]`

**0.15.324.4.9 `clearOrigins()`** `void smtrat::ModuleInput::clearOrigins ( iterator _formula ) [inline]`

**0.15.324.4.10 `contains()`** `bool smtrat::ModuleInput::contains ( const FormulaT & _subformula ) const [inline]`

**Returns**

|                          |                                                                          |
|--------------------------|--------------------------------------------------------------------------|
| <code>_subformula</code> | The formula for which to check whether it is one of the stored formulas. |
|--------------------------|--------------------------------------------------------------------------|

**Returns**

true, if the given formula is one of the stored formulas; false, otherwise.

**0.15.324.4.11 `containsBitVectorConstraints()`** `bool smtrat::ModuleInput::containsBitVectorConstraints ( ) const [inline]`

**Returns**

true, if this formula contains bit vector constraints; false, otherwise.

**0.15.324.4.12 `containsBooleanVariables()`** `bool smtrat::ModuleInput::containsBooleanVariables ( ) const [inline]`

**Returns**

true, if this formula contains Boolean variables; false, otherwise.

**0.15.324.4.13 `containsIntegerVariables()`** `bool smtrat::ModuleInput::containsIntegerVariables ( ) const [inline]`

**Returns**

true, if this formula contains Boolean variables; false, otherwise.

**0.15.324.4.14 `containsRealVariables()`** `bool smtrat::ModuleInput::containsRealVariables ( ) const [inline]`

**Returns**

true, if this formula contains Boolean variables; false, otherwise.

**0.15.324.4.15 containsUninterpretedEquations()** `bool smtrat::ModuleInput::containsUninterpretedEquations () const [inline]`

**Returns**

true, if this formula contains uninterpreted equations; false, otherwise.

**0.15.324.4.16 empty()** `bool smtrat::ModuleInput::empty () const [inline]`

**0.15.324.4.17 end() [1/2]** `iterator smtrat::ModuleInput::end () [inline]`

**0.15.324.4.18 end() [2/2]** `const_iterator smtrat::ModuleInput::end () const [inline]`

**0.15.324.4.19 erase()** `ModuleInput::iterator smtrat::ModuleInput::erase ( iterator _formula )`

**0.15.324.4.20 find() [1/4]** `ModuleInput::iterator smtrat::ModuleInput::find ( const FormulaT & _formula )`

**0.15.324.4.21 find() [2/4]** `ModuleInput::const_iterator smtrat::ModuleInput::find ( const FormulaT & _formula ) const`

**0.15.324.4.22 find() [3/4]** `ModuleInput::iterator smtrat::ModuleInput::find ( const_iterator _hint, const FormulaT & _formula )`

**0.15.324.4.23 find() [4/4]** `ModuleInput::const_iterator smtrat::ModuleInput::find ( const_iterator _hint, const FormulaT & _formula ) const`

**0.15.324.4.24 gatherVariables()** `void smtrat::ModuleInput::gatherVariables ( carl::carlVariables & vars ) const [inline]`

**0.15.324.4.25 is\_constraint\_conjunction()** `bool smtrat::ModuleInput::is_constraint_conjunction () const [inline]`

**Returns**

true, if this formula is a conjunction of constraints; false, otherwise.

**0.15.324.4.26 is\_integer\_constraint\_conjunction()** `bool smtrat::ModuleInput::is_integer_constraint_conjunction () const [inline]`

**Returns**

true, if this formula is a conjunction of integer constraints; false, otherwise.

**0.15.324.4.27 `is_only_propositional()`** `bool smtrat::ModuleInput::is_only_propositional () const [inline]`

**Returns**

true, if this formula is propositional; false, otherwise.

**0.15.324.4.28 `is_real_constraint_conjunction()`** `bool smtrat::ModuleInput::is_real_constraint_conjunction () const [inline]`

**Returns**

true, if this formula is a conjunction of real constraints; false, otherwise.

**0.15.324.4.29 `isConstraintLiteralConjunction()`** `bool smtrat::ModuleInput::isConstraintLiteralConjunction () const [inline]`

**Returns**

true, if this formula is a conjunction of literals of constraints; false, otherwise.

**0.15.324.4.30 `isIntegerConstraintLiteralConjunction()`** `bool smtrat::ModuleInput::isIntegerConstraintLiteralConjunction () const [inline]`

**Returns**

true, if this formula is a conjunction of literals of integer constraints; false, otherwise.

**0.15.324.4.31 `isRealConstraintLiteralConjunction()`** `bool smtrat::ModuleInput::isRealConstraintLiteralConjunction () const [inline]`

**Returns**

true, if this formula is a conjunction of literals of real constraints; false, otherwise.

**0.15.324.4.32 `operator FormulaT()`** `smtrat::ModuleInput::operator FormulaT () const [inline], [explicit]`

**0.15.324.4.33 `properties()`** `carl::Condition smtrat::ModuleInput::properties () const [inline]`

**Returns**

All known properties of the underlying formula of this module input.

**0.15.324.4.34 `rbegin()`** `auto smtrat::ModuleInput::rbegin () const [inline]`

**0.15.324.4.35 `removeOrigin()`** `bool smtrat::ModuleInput::removeOrigin ( iterator _formula, const FormulaT & _origin )`

**0.15.324.4.36 `removeOrigins()`** `bool smtrat::ModuleInput::removeOrigins ( iterator _formula, const std::shared_ptr< FormulasT > & _origins )`

**0.15.324.4.37 `rend()`** `auto smtrat::ModuleInput::rend ( ) const [inline]`

**0.15.324.4.38 `satisfiedBy() [1/2]`** `unsigned smtrat::ModuleInput::satisfiedBy ( const Model & _assignment ) const`

**Parameters**

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>_assignment</code> | The model to check conjunction of the stored formulas against. |
|--------------------------|----------------------------------------------------------------|

**Returns**

1, if the conjunction of the stored formulas is satisfied by the given model; 0, if the given model conflicts the conjunction of the stored formulas; 2, if it cannot be determined cheaply, whether the given model conflicts or satisfies the conjunction of the stored formulas.

**0.15.324.4.39 `satisfiedBy() [2/2]`** `unsigned smtrat::ModuleInput::satisfiedBy ( const RationalAssignment & _assignment ) const`

**Parameters**

|                          |                                                                     |
|--------------------------|---------------------------------------------------------------------|
| <code>_assignment</code> | The assignment to check conjunction of the stored formulas against. |
|--------------------------|---------------------------------------------------------------------|

**Returns**

1, if the conjunction of the stored formulas is satisfied by the given assignment; 0, if the given assignment conflicts the conjunction of the stored formulas; 2, if it cannot be determined cheaply, whether the given assignment conflicts or satisfies the conjunction of the stored formulas.

**0.15.324.4.40 `size()`** `size_t smtrat::ModuleInput::size ( void ) const [inline]`

**0.15.324.4.41 `updateProperties()`** `void smtrat::ModuleInput::updateProperties ( )`  
Updates all properties of the formula underlying this module input.

## 0.15.324.5 Friends And Related Function Documentation

**0.15.324.5.1 `Manager`** `friend class Manager [friend]`

**0.15.324.5.2 `Module`** `friend class Module [friend]`

## 0.15.325 `smtrat::settings::ModuleSettings` Struct Reference

#include <Settings.h>

## Public Member Functions

- `ModuleSettings ()`
- `void set_callbacks (std::function< bool(const std::string &)> callback_has, std::function< const std::string &(const std::string &)> callback_get)`
- `template<typename T>`
- `T get (const std::string &key, T default_value) const`

## Data Fields

- `std::vector< std::string > parameters`

### 0.15.325.1 Constructor & Destructor Documentation

#### 0.15.325.1.1 `ModuleSettings()` smtrat::settings::ModuleSettings::ModuleSettings ( ) [inline]

#### 0.15.325.2 Member Function Documentation

**0.15.325.2.1 `get()`** `template<typename T>`  
`T smtrat::settings::ModuleSettings::get (`  
    `const std::string & key,`  
    `T default_value ) const [inline]`

**0.15.325.2.2 `set_callbacks()`** `void smtrat::settings::ModuleSettings::set_callbacks (`  
    `std::function< bool(const std::string &)> callback_has,`  
    `std::function< const std::string &(const std::string &)> callback_get ) [inline]`

#### 0.15.325.3 Field Documentation

#### 0.15.325.3.1 `parameters` `std::vector<std::string> smtrat::settings::ModuleSettings::parameters`

### 0.15.326 smtrat::Module::ModuleStatistics Struct Reference

```
#include <Module.h>
```

## Public Member Functions

- `void start_add ()`
- `void start_check ()`
- `void start_remove ()`
- `void pause_all ()`
- `void continue_all ()`
- `void stop_add ()`
- `void stop_check ()`
- `void stop_remove ()`

#### 0.15.326.1 Member Function Documentation

**0.15.326.1.1 `continue_all()`** void smtrat::Module::ModuleStatistics::continue\_all ( ) [inline]

**0.15.326.1.2 `pause_all()`** void smtrat::Module::ModuleStatistics::pause\_all ( ) [inline]

**0.15.326.1.3 `start_add()`** void smtrat::Module::ModuleStatistics::start\_add ( ) [inline]

**0.15.326.1.4 `start_check()`** void smtrat::Module::ModuleStatistics::start\_check ( ) [inline]

**0.15.326.1.5 `start_remove()`** void smtrat::Module::ModuleStatistics::start\_remove ( ) [inline]

**0.15.326.1.6 `stop_add()`** void smtrat::Module::ModuleStatistics::stop\_add ( ) [inline]

**0.15.326.1.7 `stop_check()`** void smtrat::Module::ModuleStatistics::stop\_check ( ) [inline]

**0.15.326.1.8 `stop_remove()`** void smtrat::Module::ModuleStatistics::stop\_remove ( ) [inline]

## 0.15.327 smtrat::ModuleWrapper< M > Class Template Reference

```
#include <ModuleWrapper.h>
```

### Public Member Functions

- [ModuleWrapper \(\)](#)
- M & [get \(\) noexcept](#)
- bool [add \(const FormulaT &formula\)](#)
- void [remove \(const FormulaT &formula\)](#)
- bool [isAsserted \(const FormulaT &formula\)](#)
- bool [check \(\)](#)
- const std::vector< [FormulaSetT](#) > & [infeasibleSubsets \(\)](#)

### 0.15.327.1 Constructor & Destructor Documentation

**0.15.327.1.1 `ModuleWrapper()`** template<typename M >  
smtrat::ModuleWrapper< M >::ModuleWrapper ( ) [inline]

### 0.15.327.2 Member Function Documentation

**0.15.327.2.1 `add()`** template<typename M >  
bool smtrat::ModuleWrapper< M >::add (  
 const FormulaT & formula ) [inline]

**0.15.327.2.2 `check()`** template<typename M >  
bool smtrat::ModuleWrapper< M >::check ( ) [inline]

```
0.15.327.2.3 get() template<typename M >
M& smrat::ModuleWrapper< M >::get () [inline], [noexcept]

0.15.327.2.4 infeasibleSubsets() template<typename M >
const std::vector<FormulaSetT>& smrat::ModuleWrapper< M >::infeasibleSubsets () [inline]

0.15.327.2.5 isAsserted() template<typename M >
bool smrat::ModuleWrapper< M >::isAsserted (
 const FormulaT & formula) [inline]

0.15.327.2.6 remove() template<typename M >
void smrat::ModuleWrapper< M >::remove (
 const FormulaT & formula) [inline]
```

## 0.15.328 smrat::subtropical::Moment Struct Reference

Represents the normal vector component and the sign variable assigned to a variable in an original constraint.  
`#include <Subtropical.h>`

### Public Member Functions

- `Moment ()`

### Data Fields

- `const carl::Variable normal_vector`  
*Normal vector component of the separating hyperplane.*
- `const carl::Variable sign_variant`  
*Boolean variable representing the sign variant.*

#### 0.15.328.1 Detailed Description

Represents the normal vector component and the sign variable assigned to a variable in an original constraint.

#### 0.15.328.2 Constructor & Destructor Documentation

##### 0.15.328.2.1 Moment() smrat::subtropical::Moment::Moment ( ) [inline]

#### 0.15.328.3 Field Documentation

**0.15.328.3.1 normal\_vector** `const carl::Variable smrat::subtropical::Moment::normal_vector`  
Normal vector component of the separating hyperplane.

**0.15.328.3.2 sign\_variant** `const carl::Variable smrat::subtropical::Moment::sign_variant`  
Boolean variable representing the sign variant.

## 0.15.329 smrat::MonomialMappingByVariablePool Class Reference

`#include <MonomialMappingByVariablePool.h>`

## Public Member Functions

- const MonomialMap & **getMMonomialMapping()** const
- void **insertMonomialMapping** (carl::Variable **variable**, carl::Monomial::Arg **monomial**)
- carl::Variable **variable** (carl::Monomial::Arg **monomial**)
- bool **isNull** (carl::Variable **variable**)
- carl::Monomial::Arg **monomial** (carl::Variable **variable**)

## Friends

- class carl::Singleton< MonomialMappingByVariablePool >

### 0.15.329.1 Member Function Documentation

**0.15.329.1.1 getMMonomialMapping()** const MonomialMap& smtrat::MonomialMappingByVariablePool::getMMonomialMapping ( ) const [inline]

**0.15.329.1.2 insertMonomialMapping()** void smtrat::MonomialMappingByVariablePool::insertMonomialMapping ( carl::Variable **variable**, carl::Monomial::Arg **monomial** ) [inline]

**0.15.329.1.3 isNull()** bool smtrat::MonomialMappingByVariablePool::isNull ( carl::Variable **variable** ) [inline]

**0.15.329.1.4 monomial()** carl::Monomial::Arg smtrat::MonomialMappingByVariablePool::monomial ( carl::Variable **variable** ) [inline]

**0.15.329.1.5 variable()** carl::Variable smtrat::MonomialMappingByVariablePool::variable ( carl::Monomial::Arg **monomial** ) [inline]

### 0.15.329.2 Friends And Related Function Documentation

**0.15.329.2.1 carl::Singleton< MonomialMappingByVariablePool >** friend class carl::Singleton< MonomialMappingByVariablePool > [friend]

## 0.15.330 smtrat::expression::NaryExpression Struct Reference

```
#include <ExpressionContent.h>
```

## Public Member Functions

- NaryExpression (NaryType **\_type**, Expressions && **\_expressions**)
- NaryExpression (NaryType **\_type**, const std::initializer\_list< Expression > & **\_expressions**)
- void **normalize** ()

## Data Fields

- NaryType **type**
- Expressions **expressions**

### 0.15.330.1 Constructor & Destructor Documentation

**0.15.330.1.1 NaryExpression() [1/2]** smtrat::expression::NaryExpression::NaryExpression (

```
NaryType _type,
Expressions && _expressions) [inline]
```

**0.15.330.1.2 NaryExpression() [2/2]** smtrat::expression::NaryExpression::NaryExpression (

```
NaryType _type,
const std::initializer_list< Expression > & _expressions) [inline]
```

### 0.15.330.2 Member Function Documentation

**0.15.330.2.1 normalize()** void smtrat::expression::NaryExpression::normalize ( ) [inline]

### 0.15.330.3 Field Documentation

**0.15.330.3.1 expressions** Expressions smtrat::expression::NaryExpression::expressions

**0.15.330.3.2 type** NaryType smtrat::expression::NaryExpression::type

## 0.15.331 smtrat::expression::simplifier::NegationSimplifier Struct Reference

```
#include <NegationSimplifier.h>
```

### Public Member Functions

- const ExpressionContent \* **simplify** (const UnaryExpression &expr) const
- const ExpressionContent \* **operator()** (const carl::Variable &expr) const
- const ExpressionContent \* **operator()** (const ITEExpression &expr) const
- const ExpressionContent \* **operator()** (const QuantifierExpression &expr) const
- const ExpressionContent \* **operator()** (const UnaryExpression &expr) const
- const ExpressionContent \* **operator()** (const BinaryExpression &expr) const
- const ExpressionContent \* **operator()** (const NaryExpression &expr) const
- const ExpressionContent \* **operator()** (const ExpressionContent \*\_ec) const

### Protected Member Functions

- virtual const ExpressionContent \* **simplify** (const carl::Variable &) const
- virtual const ExpressionContent \* **simplify** (const ITEExpression &) const
- virtual const ExpressionContent \* **simplify** (const QuantifierExpression &) const
- virtual const ExpressionContent \* **simplify** (const BinaryExpression &) const
- virtual const ExpressionContent \* **simplify** (const NaryExpression &) const

### 0.15.331.1 Member Function Documentation

**0.15.331.1.1 operator() [1/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (

const `BinaryExpression & expr` ) const [inline], [inherited]

**0.15.331.1.2 operator() [2/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (

const `carl::Variable & expr` ) const [inline], [inherited]

**0.15.331.1.3 operator() [3/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (

const `ExpressionContent * _ec` ) const [inline], [inherited]

**0.15.331.1.4 operator() [4/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (

const `ITEEExpression & expr` ) const [inline], [inherited]

**0.15.331.1.5 operator() [5/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (

const `NaryExpression & expr` ) const [inline], [inherited]

**0.15.331.1.6 operator() [6/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (

const `QuantifierExpression & expr` ) const [inline], [inherited]

**0.15.331.1.7 operator() [7/7]** const `ExpressionContent*` smtrat::expression::simplifier::Base<→ Simplifier::operator() (

const `UnaryExpression & expr` ) const [inline], [inherited]

**0.15.331.1.8 simplify() [1/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (

const `BinaryExpression &`  ) const [inline], [protected], [virtual], [inherited]

**0.15.331.1.9 simplify() [2/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (

const `carl::Variable &`  ) const [inline], [protected], [virtual], [inherited]

**0.15.331.1.10 simplify() [3/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (

const `ITEEExpression &`  ) const [inline], [protected], [virtual], [inherited]

**0.15.331.1.11 simplify() [4/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier::BaseSimplifier::simplify (

const `NaryExpression &`  ) const [inline], [protected], [virtual], [inherited]

Reimplemented in `smtrat::expression::simplifier::SingletonSimplifier`, `smtrat::expression::simplifier::MergeSimplifier`, and `smtrat::expression::simplifier::DuplicateSimplifier`.

**0.15.331.1.12 `simplify()` [5/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier $\leftarrow$  ::BaseSimplifier::simplify ( const `QuantifierExpression` & ) const [inline], [protected], [virtual], [inherited]

**0.15.331.1.13 `simplify()` [6/6]** const `ExpressionContent*` smtrat::expression::simplifier::Negation $\leftarrow$  Simplifier::simplify ( const `UnaryExpression` & expr ) const [inline], [virtual]  
Reimplemented from `smtrat::expression::simplifier::BaseSimplifier`.

## 0.15.332 `smtrat::NewCADModule`< `Settings` > Class Template Reference

```
#include <NewCADModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
- `NewCADModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)`
- `~NewCADModule ()`
- `bool informCore (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- `void init ()`

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `bool addCore (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- `void removeCore (ModuleInput::const_iterator _subformula)`

*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- `void updateModel () const`

*Updates the current assignment into the model.*
- `Answer checkCore ()`

*Checks the received formula for consistency.*
- `bool inform (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- `void deform (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- `bool add (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- `virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`

*Checks the received formula for consistency.*
- `virtual void remove (ModuleInput::const_iterator _subformula)`

*Removes everything related to the given sub-formula of the received formula.*
- `virtual void updateAllModels ()`

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`

- `Answer solverState () const`
- `std::size_t id () const`
- `void setId (std::size_t _id)`

*Sets this module's unique ID to identify itself.*
- `thread_priority threadPriority () const`
- `void setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- `const ModuleInput * pReceivedFormula () const`
- `const ModuleInput & rReceivedFormula () const`
- `const ModuleInput * pPassedFormula () const`
- `const ModuleInput & rPassedFormula () const`
- `const Model & model () const`
- `const std::vector< Model > & allModels () const`
- `const std::vector< FormulaSetT > & infeasibleSubsets () const`
- `const std::vector< Module * > & usedBackends () const`
- `const carl::FastSet< FormulaT > & constraintsToInform () const`
- `const carl::FastSet< FormulaT > & informedConstraints () const`
- `void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- `bool hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- `void clearLemmas ()`

*Deletes all yet found lemmas.*
- `const std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- `void receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- `const smtrat::Conditionals & answerFound () const`
- `bool isPreprocessor () const`
- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- `void updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- `virtual std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- `void printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- `void printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*

- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const  
*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector<`Branching`> `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector<`FormulaT`> `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual void `deinformCore` (const `FormulaT` &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair<`ModuleInput::iterator`, bool> `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Removes the origin of the given formula from the passed formula.*
- std::pair<`ModuleInput::iterator`, bool> `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr<std::vector<`FormulaT`>> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*

- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const `Model` & `backendsModel` () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel` () const
- void `getBackendsAllModels` () const
 

*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends` (bool \_final, bool \_full, carl::Variable \_objective)
 

*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends` ()
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula` (`ModuleInput::iterator` \_subformula, bool \_ignoreOrigins=false)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula` ()
- std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const
 

*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &←\_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*

- void `splitUnequalConstraint` (const `FormulaT` &)
 

*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &`formula`)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
 

*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)
 

*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &`_modelA`, const `Model` &`_modelB`)
 

*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`

*A reference to the manager.*
- `Model` `mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`

*Stores all satisfying assignments.*
- bool `mModelComputed`

*True, if the model has already been computed.*
- bool `mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals `mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`

*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput`::iterator `mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.332.1 Member Typedef Documentation

**0.15.332.1.1 SettingsType** `template<typename Settings >`  
`typedef smtrat::NewCADModule< Settings >::SettingsType`

### 0.15.332.2 Member Enumeration Documentation

**0.15.332.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.332.3 Constructor & Destructor Documentation

**0.15.332.3.1 NewCADModule()** `template<typename Settings >`  
`smtrat::NewCADModule< Settings >::NewCADModule (`  
`const ModuleInput * _formula,`  
`Conditionals & _conditionals,`  
`Manager * _manager = nullptr )`

**0.15.332.3.2 ~NewCADModule()** `template<typename Settings >`  
`smtrat::NewCADModule< Settings >::~NewCADModule ( )`

### 0.15.332.4 Member Function Documentation

**0.15.332.4.1 add()** `bool smtrat::Module::add (`  
`ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.332.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`

```
const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.332.4.3 addCore()** `template<typename Settings >`

```
bool smtrat::NewCADModule< Settings >::addCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.332.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula`

```
(
```

```
const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.332.4.5 addLemma()** `void smtrat::Module::addLemma (`

```
const FormulaT & _lemma,
```

```
const LemmaType & _lt = LemmaType::NORMAL,
```

```
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
```

```
[inherited]
```

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.332.4.6 addOrigin()** `void smtrat::Module::addOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.332.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.332.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.332.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.332.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>**

```
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.332.4.11 allModels() const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]****Returns**

All satisfying assignments, if existent.

**0.15.332.4.12 anAnswerFound() bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.332.4.13 answerFound() const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.332.4.14 backendsModel() const Model & smtrat::Module::backendsModel () const [protected], [inherited]**

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.332.4.15 branchAt() [1/4]** bool smrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.332.4.16 branchAt() [2/4]** bool smrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.332.4.17 branchAt() [3/4]** bool smrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.332.4.18 branchAt() [4/4]** bool smrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.332.4.19 check() Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

```
0.15.332.4.20 checkCore() template<typename Settings >
Answer smtrat::NewCADModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.332.4.21 checkInfSubsetForMinimality() void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infsSubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infsSubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.332.4.22 checkModel()** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.332.4.23 cleanModel()** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.332.4.24 clearLemmas()** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.332.4.25 clearModel()** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.332.4.26 clearModels()** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.332.4.27 clearPassedFormula()** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.332.4.28 collectOrigins() [1/2]** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.332.4.29 collectOrigins() [2/2]** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.332.4.30 collectTheoryPropagations()** `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

**0.15.332.4.31 constraintsToInform()** const carl::FastSet<[FormulaT](#)>& smrat::Module::constraintsToInform( ) const [inline], [inherited]

#### Returns

The constraints which the backends must be informed about.

**0.15.332.4.32 currentlySatisfied()** virtual unsigned smrat::Module::currentlySatisfied( const [FormulaT](#) & ) const [inline], [virtual], [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), and [smrat::LRAModule< LRASettings >](#)

**0.15.332.4.33 currentlySatisfiedByBackend()** unsigned smrat::Module::currentlySatisfiedByBackend( const [FormulaT](#) & \_formula ) const [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.332.4.34 deinform()** void smrat::Module::deinform( const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.332.4.35 deinformCore()** virtual void smrat::Module::deinformCore( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), and [smrat::LRAModule< LRASettings >](#)

**0.15.332.4.36 determine\_smallest\_origin()** size\_t smrat::Module::determine\_smallest\_origin( const std::vector< [FormulaT](#) > & origins ) const [protected], [inherited]

**Parameters**

|                |                              |
|----------------|------------------------------|
| <i>origins</i> | A vector of sets of origins. |
|----------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.332.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::erase<SubformulaFromPassedFormula (`

`ModuleInput::iterator _subformula,`  
`bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                       |                                                                                                              |
|-----------------------|--------------------------------------------------------------------------------------------------------------|
| <i>_subformula</i>    | The sub-formula to remove from the passed formula.                                                           |
| <i>_ignoreOrigins</i> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.332.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceived<VariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.332.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::find<BestOrigin (`

`const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

**0.15.332.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformula<ToPass ( ) const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.332.4.41 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

#### Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.332.4.42 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.332.4.43 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.332.4.44 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.332.4.45 getBackendsModel()** `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.332.4.46 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.332.4.47 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (`

`const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.332.4.48 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.332.4.49 getModelEqualities()** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.332.4.50 getOrigins() [1/3]** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulaSetT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.332.4.51 getOrigins() [2/3]** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulasT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.332.4.52 getOrigins() [3/3]** `const FormulaT& smtrat::Module::getOrigins (`

```
ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.332.4.53 getReceivedFormulaSimplified()** `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.332.4.54 hasLemmas()** `bool smtrat::Module::hasLemmas () [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.332.4.55 hasValidInfeasibleSubset()** `bool smtrat::Module::hasValidInfeasibleSubset () const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.332.4.56 id()** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.332.4.57 infeasibleSubsets()** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.332.4.58 inform()** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.332.4.59 informBackends()** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.332.4.60 informCore()** `template<typename Settings > bool smtrat::NewCADModule< Settings >::informCore (`

```
 const FormulaT & _constraint) [virtual]
```

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

`false`, if it can be easily decided whether the given constraint is inconsistent; `true`, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.332.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`

#### Returns

The position of the first constraint of which no backend has been informed about.

**0.15.332.4.62 init()** `template<typename Settings > void smtrat::NewCADModule< Settings >::init ( ) [virtual]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not and will not be called more than once and only before the first runBackends call.

Reimplemented from [smtrat::Module](#).

**0.15.332.4.63 is\_minimizing()** `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`

**0.15.332.4.64 isLemmaLevel()** `bool smtrat::Module::isLemmaLevel (`

```
 LemmaLevel level) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.332.4.65 isPreprocessor()** `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`

#### Returns

`true`, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.332.4.66 lemmas()** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.332.4.67 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & [\\_vecSetA](#), const std::vector< [FormulaT](#) > & [\\_vecSetB](#) ) const [protected], [inherited]  
Merges the two vectors of sets into the first one.  
({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                          |                                  |
|--------------------------|----------------------------------|
| <a href="#">_vecSetA</a> | A vector of sets of constraints. |
| <a href="#">_vecSetB</a> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.332.4.68 model()** const [Model&](#) smtrat::Module::model ( ) const [inline], [inherited]

**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.332.4.69 modelsDisjoint()** bool smtrat::Module::modelsDisjoint ( const [Model](#) & [\\_modelA](#), const [Model](#) & [\\_modelB](#) ) [static], [protected], [inherited]  
Checks whether there is no variable assigned by both models.

**Parameters**

|                         |                                |
|-------------------------|--------------------------------|
| <a href="#">_modelA</a> | The first model to check for.  |
| <a href="#">_modelB</a> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.332.4.70 moduleName()** template<typename Settings > std::string smtrat::NewCADModule< [Settings](#) >::moduleName ( ) const [inline], [virtual]

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.332.4.71 objective()** carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]

**0.15.332.4.72 originInReceivedFormula()** `bool smtrat::Module::originInReceivedFormula (`  
    `const FormulaT & _origin ) const [protected], [inherited]`

**0.15.332.4.73 passedFormulaBegin()** `ModuleInput::iterator smtrat::Module::passedFormulaBegin (`  
    `) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.332.4.74 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
[inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.332.4.75 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.332.4.76 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.332.4.77 print()** `void smtrat::Module::print (`  
    `const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.332.4.78 printAllModels()** `void smtrat::Module::printAllModels (`  
    `std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.332.4.79 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the infeasible subsets.

#### Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.332.4.80 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

#### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.332.4.81 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

#### Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.332.4.82 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

#### Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.332.4.83 probablyLooping()** bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & *\_branchingPolynomial*, const Rational & *\_branchingValue* ) const [protected], [inherited]

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.332.4.84 receivedFormulaChecked()** void smtrat::Module::receivedFormulaChecked () [inline], [inherited]  
Notifies that the received formulas has been checked.

**0.15.332.4.85 receivedFormulasAsInfeasibleSubset()** void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const\_iterator \_subformula ) [inline], [protected], [inherited]  
Stores an infeasible subset consisting only of the given received formula.

**0.15.332.4.86 receivedVariable()** bool smtrat::Module::receivedVariable ( carl::Variable::Arg \_var ) const [inline], [inherited]

**0.15.332.4.87 remove()** void smtrat::Module::remove ( ModuleInput::const\_iterator \_subformula ) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.  
However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub formula of the received formula to remove. |
|-------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.332.4.88 removeCore()** template<typename Settings >  
void smtrat::NewCADModule< Settings >::removeCore ( ModuleInput::const\_iterator \_subformula ) [virtual]

Removes the subformula of the received formula at the given position to the considered ones of this module.  
Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|             |                                           |
|-------------|-------------------------------------------|
| _subformula | The position of the subformula to remove. |
|-------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.332.4.89 removeOrigin()** std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin ( ModuleInput::iterator \_formula, const FormulaT & \_origin ) [inline], [protected], [inherited]

**0.15.332.4.90 removeOrigins()** std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins ( ModuleInput::iterator \_formula, const std::shared\_ptr< std::vector< FormulaT >> & \_origins ) [inline], [protected], [inherited]

**0.15.332.4.91 rPassedFormula()** const ModuleInput& smtrat::Module::rPassedFormula () const

[inline], [inherited]

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.332.4.92 rReceivedFormula()** `const ModuleInput& smtrat::Module::rReceivedFormula () const`  
 [inline], [inherited]

#### Returns

A reference to the formula passed to this module.

**0.15.332.4.93 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`  
 Reimplemented in [smtrat::PModule](#).

**0.15.332.4.94 runBackends() [2/2]** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.332.4.95 setId()** `void smtrat::Module::setId (`  
`std::size_t _id ) [inline], [inherited]`

Sets this modules unique ID to identify itself.

#### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <code>id</code> | The id to set this module's id to. |
|-----------------|------------------------------------|

**0.15.332.4.96 setThreadPriority()** `void smtrat::Module::setThreadPriority (`  
`thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.332.4.97 `solverState()`** Answer smtrat::Module::solverState ( ) const [inline], [inherited]**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.332.4.98 `splitUnequalConstraint()`** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]

Adds the following lemmas for the given constraint  $p \neq 0$ .

$$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.332.4.99 `threadPriority()`** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.332.4.100 `updateAllModels()`** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.332.4.101 `updateLemmas()`** void smtrat::Module::updateLemmas ( ) [inherited]

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.332.4.102 `updateModel()`** template<typename Settings > void smtrat::NewCADModule< Settings >::updateModel ( ) const [virtual]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.332.4.103 `usedBackends()`** const std::vector<Module\*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.332.5 Field Documentation

**0.15.332.5.1 mAllBackends** std::vector<[Module](#)\*> smrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

**0.15.332.5.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.332.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer [protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.332.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.332.5.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.332.5.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.332.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.332.5.8 mFirstUncheckedReceivedSubformula** [ModuleInput](#)::const\_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.332.5.9 mFoundAnswer** [Conditionals](#) smrat::Module::mFoundAnswer [protected], [inherited]

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.332.5.10 mFullCheck** bool smrat::Module::mFullCheck [protected], [inherited]

false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.332.5.11 mInfeasibleSubsets** std::vector<[FormulaSetT](#)> smrat::Module::mInfeasibleSubsets [protected], [inherited]

Stores the infeasible subsets.

**0.15.332.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.332.5.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.332.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.332.5.15 mModel** `Model smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.332.5.16 mModelComputed** `bool smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.332.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.332.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.332.5.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.332.5.20 mpManager** `Manager* const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.332.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter` [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.332.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.332.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations` [protected], [inherited]

**0.15.332.5.24 mUsedBackends** std::vector<Module\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.332.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.333 smtrat::NewCoveringModule< Settings > Class Template Reference

```
#include <NewCoveringModule.h>
```

### Public Types

- using `SettingsType = Settings`
- enum class `LemmaType` : unsigned { `NORMAL` = 0 , `PERMANENT` = 1 }

### Public Member Functions

- `NewCoveringModule` (const `ModuleInput` \*\_formula, `Conditionals` &\_conditionals, `Manager` \*\_manager= nullptr)
 

*Informs the module about the given constraint.*
- `~NewCoveringModule` ()
 

*The module has to take the given sub-formula of the received formula into account.*
- `bool informCore` (const `FormulaT` &\_constraint)
 

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `bool addCore` (`ModuleInput::const_iterator` \_subformula)
 

*Updates the current assignment into the model.*
- `void removeCore` (`ModuleInput::const_iterator` \_subformula)
 

*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- `void updateModel` () const
 

*Updates the current assignment into the model.*
- `Answer checkCore` ()
 

*Processes the current answer, i.e.*
- `void processAnswer` ()
 

*Adds the given constraint to the backend.*
- `size_t addConstraintsSAT` ()
 

*Removes the given constraints from the backend.*
- `void removeConstraintsSAT` ()
 

*Adds the given constraint to the backend.*
- `void addConstraintsUNSAT` ()
 

*Removes the given constraints from the backend Also removes all stored information about the removed constraints.*
- `std::optional< Answer > doBacktracking` ()
 

*The actual backtracking algorithm, which is called when we have constraints to add and no constraints to remove.*
- `std::optional< Answer > doIncremental` ()
 

*The actual incremental algorithm, which is called when we have constraints to add and no constraints to remove.*
- `std::optional< Answer > doIncrementalAndBacktracking` ()
 

*Algorithms which contains backtracking and incremental checking, its called when we have constraints to add and constraints to remove.*
- `bool inform` (const `FormulaT` &\_constraint)

- **Informs the module about the given constraint.**
- void **deinform** (const **FormulaT** &\_constraint)
 

*The inverse of informing about a constraint.*
- bool **add** (**ModuleInput::const\_iterator** \_subformula)
 

*The module has to take the given sub-formula of the received formula into account.*
- virtual **Answer check** (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)
 

*Checks the received formula for consistency.*
- virtual void **remove** (**ModuleInput::const\_iterator** \_subformula)
 

*Removes everything related to the given sub-formula of the received formula.*
- virtual void **updateAllModels** ()
 

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const
 

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned **currentlySatisfiedByBackend** (const **FormulaT** &\_formula) const
- virtual unsigned **currentlySatisfied** (const **FormulaT** &) const
- bool **receivedVariable** (carl::Variable::Arg \_var) const
- **Answer solverState** () const
- std::size\_t **setId** (std::size\_t \_id)
 

*Sets this modules unique ID to identify itself.*
- **thread\_priority threadPriority** () const
- void **setThreadPriority** (**thread\_priority** \_threadPriority)
 

*Sets the priority of this module to get a thread for running its check procedure.*
- const **ModuleInput** \* **pReceivedFormula** () const
- const **ModuleInput** & **rReceivedFormula** () const
- const **ModuleInput** \* **pPassedFormula** () const
- const **ModuleInput** & **rPassedFormula** () const
- const **Model** & **model** () const
- const std::vector< **Model** > & **allModels** () const
- const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
- const std::vector< **Module** \* > & **usedBackends** () const
- const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
- const carl::FastSet< **FormulaT** > & **informedConstraints** () const
- void **addLemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lt=LemmaType::NORMAL, const **FormulaT** &\_preferredFormula=**FormulaT**(carl::FormulaType::TRUE))
 

*Stores a lemma being a valid formula.*
- bool **hasLemmas** ()
 

*Checks whether this module or any of its backends provides any lemmas.*
- void **clearLemmas** ()
 

*Deletes all yet found lemmas.*
- const std::vector< **Lemma** > & **lemmas** () const
- **ModuleInput::const\_iterator** **firstUncheckedReceivedSubformula** () const
- **ModuleInput::const\_iterator** **firstSubformulaToPass** () const
- void **receivedFormulaChecked** ()
 

*Notifies that the received formulas has been checked.*
- const **smrat::Conditionals** & **answerFound** () const
- bool **isPreprocessor** () const
- virtual std::string **moduleName** () const
- carl::Variable **objective** () const
- bool **is\_minimizing** () const
- void **excludeNotReceivedVariablesFromModel** () const
 

*Excludes all variables from the current model, which do not occur in the received formula.*

- void `updateLemmas ()`  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `isValidInfeasibleSubset () const`
- void `checkInfeasibleSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infssubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, FormulaT > `getReceivedFormulaSimplified ()`
- void `print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore = 5`  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches = 0`  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual void `deinformCore (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- bool `anAnswerFound () const`  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel () const`  
*Clears the assignment, if any was found.*
- void `clearModels () const`  
*Clears all assignments, if any was found.*

- void `cleanModel () const`  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin ()`
- `ModuleInput::iterator passedFormulaEnd ()`
- void `addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
- void `getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- std::pair< `ModuleInput::iterator, bool` > `removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
- std::pair< `ModuleInput::iterator, bool` > `removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
- void `informBackends (const FormulaT &_constraint)`  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform (const FormulaT &_constraint)`  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator, bool` > `addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula (const FormulaT &_origin) const`
- std::pair< `ModuleInput::iterator, bool` > `addSubformulaToPassedFormula (const FormulaT &_formula)`  
*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator, bool` > `addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`  
*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator, bool` > `addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`  
*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset ()`  
*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`  
*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin (const std::vector< FormulaT > &_origins) const`
- void `getInfeasibleSubsets ()`  
*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets (const Module &backend) const`  
*Get the infeasible subsets the given backend provides.*
- const `Model & backendsModel () const`  
*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel () const`
- void `getBackendsAllModels () const`  
*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`  
*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`
- std::vector< `FormulaT` > `merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`

- `size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const`

*Merges the two vectors of sets into the first one.*
- `bool probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const`

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- `bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- `bool branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &&_premise=std::vector< FormulaT >())`
- `bool branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`
- `bool branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())`
- `void splitUnequalConstraint (const FormulaT &)`

*Adds the following lemmas for the given constraint p!=0.*
- `unsigned checkModel () const`
- `void addInformationRelevantFormula (const FormulaT &formula)`

*Adds a formula to the InformationRelevantFormula.*
- `const std::set< FormulaT > & getInformationRelevantFormulas ()`

*Gets all InformationRelevantFormulas.*
- `bool isLemmaLevel (LemmaLevel level)`

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- `static bool modelsDisjoint (const Model &_modelA, const Model &_modelB)`

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector< FormulaSetT > mInfeasibleSubsets`

*Stores the infeasible subsets.*
- `Manager *const mpManager`

*A reference to the manager.*
- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*
- `std::vector< Model > mAllModels`

*Stores all satisfying assignments.*
- `bool mModelComputed`

*True, if the model has already been computed.*
- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- `std::vector< TheoryPropagation > mTheoryPropagations`
- `std::atomic< Answer > mSolverState`

- States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* mBackendsFoundAnswer  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
  - Conditionals mFoundAnswer  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
  - std::vector< Module \* > mUsedBackends  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
  - std::vector< Module \* > mAllBackends  
*The backends of this module which have been used.*
  - std::vector< Lemma > mLemmas  
*Stores the lemmas being valid formulas this module or its backends made.*
  - ModuleInput::iterator mFirstSubformulaToPass  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
  - carl::FastSet< FormulaT > mConstraintsToInform  
*Stores the constraints which the backends must be informed about.*
  - carl::FastSet< FormulaT > mInformedConstraints  
*Stores the position of the first constraint of which no backend has been informed about.*
  - ModuleInput::const\_iterator mFirstUncheckedReceivedSubformula  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
  - unsigned mSmallerMusesCheckCounter  
*Counter used for the generation of the smt2 files to check for smaller muses.*
  - std::vector< std::size\_t > mVariableCounters  
*Maps variables to the number of their occurrences.*

### 0.15.333.1 Member Typedef Documentation

```
0.15.333.1.1 SettingsType template<typename Settings >
using smtrat::NewCoveringModule< Settings >::SettingsType = Settings
```

### 0.15.333.2 Member Enumeration Documentation

```
0.15.333.2.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.333.3 Constructor & Destructor Documentation

```
0.15.333.3.1 NewCoveringModule() template<typename Settings >
smtrat::NewCoveringModule< Settings >::NewCoveringModule (
 const ModuleInput * _formula,
 Conditionals & _conditionals,
 Manager * _manager = nullptr)
```

---

**0.15.333.3.2 ~NewCoveringModule()** template<typename Settings >  
`smtrat::NewCoveringModule< Settings >::~NewCoveringModule ( )`

#### 0.15.333.4 Member Function Documentation

**0.15.333.4.1 add()** bool smtrat::Module::add (  
`ModuleInput::const_iterator _subformula`) [inherited]

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.333.4.2 addConstraintsSAT()** template<typename Settings >  
`size_t smtrat::NewCoveringModule< Settings >::addConstraintsSAT ( )`  
 Adds the given constraint to the backend.

##### Returns

the lowest level with an unsatisfied new constraint

##### Note

returns mVariableOrdering.size() + 1 if all new constraints are satisfied

If not all new constraints are satisfied the

**0.15.333.4.3 addConstraintsUNSAT()** template<typename Settings >  
`void smtrat::NewCoveringModule< Settings >::addConstraintsUNSAT ( )`  
 Adds the given constraint to the backend.

**0.15.333.4.4 addConstraintToInform()** void smtrat::Module::addConstraintToInform (  
`const FormulaT & _constraint`) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

##### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in `smtrat::SATModule< Settings >`.

**0.15.333.4.5 addCore()** template<typename Settings >  
`bool smtrat::NewCoveringModule< Settings >::addCore (`  
`ModuleInput::const_iterator _subformula`) [virtual]

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.333.4.6 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula (`  
`const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.333.4.7 addLemma()** `void smtrat::Module::addLemma (`  
`const FormulaT & _lemma,`  
`const LemmaType & _lt = LemmaType::NORMAL,`  
`const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],`  
[inherited]

Stores a lemma being a valid formula.

#### Parameters

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.333.4.8 addOrigin()** `void smtrat::Module::addOrigin (`  
`ModuleInput::iterator _formula,`  
`const FormulaT & _origin ) [inline], [protected], [inherited]`

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.333.4.9 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator, bool> smtrat::Module::addReceivedSubformulaToPassedFormula (`  
`ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

#### 0.15.333.4.10 `addSubformulaToPassedFormula()` [1/3] `std::pair<ModuleInput::iterator,bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's `removeSubformula`, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

#### 0.15.333.4.11 `addSubformulaToPassedFormula()` [2/3] `std::pair<ModuleInput::iterator,bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

#### 0.15.333.4.12 `addSubformulaToPassedFormula()` [3/3] `std::pair<ModuleInput::iterator,bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.333.4.13 `allModels()`** `const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]`

**Returns**

All satisfying assignments, if existent.

**0.15.333.4.14 `anAnswerFound()`** `bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.333.4.15 `answerFound()`** `const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]`

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.333.4.16 `backendsModel()`** `const Model & smtrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.333.4.17 `branchAt() [1/4]`** `bool smtrat::Module::branchAt ( carl::Variable::Arg _var, const Rational & _value, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & _premise = std::vector< FormulaT > () ) [inline], [protected], [inherited]`

```
0.15.333.4.18 branchAt() [2/4] bool smtrat::Module::branchAt (
 carl::Variable::Arg _var,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

```
0.15.333.4.19 branchAt() [3/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

```
0.15.333.4.20 branchAt() [4/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.333.4.21 check() Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.333.4.22 `checkCore()`** `template<typename Settings >`  
`Answer smtrat::NewCoveringModule< Settings >::checkCore ( ) [virtual]`  
 Checks the received formula for consistency.

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.333.4.23 `checkInfSubsetForMinimality()`** `void smtrat::Module::checkInfSubsetForMinimality (`  
`std::vector< FormulaSetT >::const_iterator _infsSubset,`  
`const std::string & _filename = "smaller_muses",`  
`unsigned _maxSizeDifference = 1 ) const [inherited]`

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.333.4.24 `checkModel()`** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.333.4.25 `cleanModel()`** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.333.4.26 `clearLemmas()`** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

**0.15.333.4.27 `clearModel()`** `void smtrat::Module::clearModel () const [inline], [protected], [inherited]`  
 Clears the assignment, if any was found.

**0.15.333.4.28 `clearModels()`** `void smtrat::Module::clearModels () const [inline], [protected], [inherited]`  
 Clears all assignments, if any was found.

**0.15.333.4.29 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula () [protected], [inherited]`

**0.15.333.4.30 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins (const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.333.4.31 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins (const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.333.4.32 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations () [inherited]`

**0.15.333.4.33 `constraintsToInform()`** `const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]`

#### Returns

The constraints which the backends must be informed about.

**0.15.333.4.34 `currentlySatisfied()`** `virtual unsigned smtrat::Module::currentlySatisfied (const FormulaT & ) const [inline], [virtual], [inherited]`

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, and `smtrat::LRAModule< LRASettings >`

**0.15.333.4.35 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.333.4.36 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.333.4.37 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASetting >](#)

**0.15.333.4.38 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.333.4.39 doBacktracking()** `template<typename Settings >`

`std::optional<Answer> smtrat::NewCoveringModule< Settings >::doBacktracking ( )`

The actual backtracking algorithm, which is called when we have constraints to add and no constraints to remove.

Removes all cells which were created as a rest of the constraints which are to be removed if level 0 is still fully covered after removing the constraints UNSAT is returned Otherwise std::nullopt is returned and level 1 is newly sampled and the algorithm starts again from level 1

**Returns**

`std::optional<Answer>`, which contains the answer if it can be deduced directly

**0.15.333.4.40 doIncremental()** template<typename Settings >  
 std::optional<Answer> smtrat::NewCoveringModule< Settings >::doIncremental ( )  
 The actual incremental algorithm, which is called when we have constraints to add and no constraints to remove.  
 Tests the new constraints with the last satisfying assignment, if the new constraints are also satisfied. If the new constraints are all satisfied SAT is returned accordingly Otherwise std::nullopt is returned and the stored information from the lowest level with an unsatisfied new constraint is removed

**Returns**

std::optional<Answer>, which contains the answer if it can be deduced directly

**0.15.333.4.41 doIncrementalAndBacktracking()** template<typename Settings >  
 std::optional<Answer> smtrat::NewCoveringModule< Settings >::doIncrementalAndBacktracking ( )  
 Algorithms which contains backtracking and incremental checking, its called when we have constraints to add and constraints to remove.

**Returns**

std::optional<Answer>, which contains the answer if it can be deduced directly

**0.15.333.4.42 eraseSubformulaFromPassedFormula()** ModuleInput::iterator smtrat::Module::erase<  
 SubformulaFromPassedFormula (   
     ModuleInput::iterator \_subformula,  
     bool \_ignoreOrigins = false ) [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
 Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in smtrat::ICPModule< Settings >, smtrat::ICPModule< ICPSettings4 >, and smtrat::ICPModule< ICPSettings1 >.

**0.15.333.4.43 excludeNotReceivedVariablesFromModel()** void smtrat::Module::excludeNotReceived<  
 VariablesFromModel ( ) const [inherited]  
 Excludes all variables from the current model, which do not occur in the received formula.

**0.15.333.4.44 findBestOrigin()** std::vector< FormulaT >::const\_iterator smtrat::Module::find<  
 BestOrigin (   
     const std::vector< FormulaT > & \_origins ) const [protected], [inherited]

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

Returns

**0.15.333.4.45 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.333.4.46 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.333.4.47 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.333.4.48 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.333.4.49 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.333.4.50 getBackendsModel()** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.333.4.51 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.333.4.52 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.333.4.53 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.333.4.54 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.333.4.55 `getOrigins() [1/3]`** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.333.4.56 `getOrigins() [2/3]`** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.333.4.57 `getOrigins() [3/3]`** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.333.4.58 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified( ) [virtual], [inherited]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.333.4.59 `hasLemmas()`** `bool smtrat::Module::hasLemmas( ) [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.333.4.60 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.333.4.61 `id()`** `std::size_t smtrat::Module::id( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.333.4.62 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.333.4.63 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.333.4.64 informBackends()** `void smrat::Module::informBackends (`

`const FormulaT & _constraint ) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.333.4.65 informCore()** `template<typename Settings >`

`bool smrat::NewCoveringModule< Settings >::informCore (`

`const FormulaT & _constraint ) [virtual]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.333.4.66 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informed→`

`Constraints ( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.333.4.67 init()** `template<typename Settings >`

`void smrat::NewCoveringModule< Settings >::init ( ) [virtual]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not and will not be called more than once and only before the first runBackends call.

Reimplemented from [smrat::Module](#).

**0.15.333.4.68 is\_minimizing()** `bool smrat::Module::is_minimizing ( ) const [inline], [inherited]`**0.15.333.4.69 isLemmaLevel()** `bool smrat::Module::isLemmaLevel (`

`LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.333.4.70 `isPreprocessor()`** `bool smtrat::Module::isPreprocessor () const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.333.4.71 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas () const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.333.4.72 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.333.4.73 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.333.4.74 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                |                                |
|----------------|--------------------------------|
| <i>_modelA</i> | The first model to check for.  |
| <i>_modelB</i> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.333.4.75 `moduleName()`** `virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]`

**Returns**

The name of the given module type as name.

Reimplemented in `smtrat::VSModule< Settings >`, `smtrat::UnionFindModule< Settings >`, `smtrat::UFCegarModule< Settings >`, `smtrat::STropModule< Settings >`, `smtrat::SplitSOSModule< Settings >`, `smtrat::PBPPModule< Settings >`, `smtrat::PBGaussModule< Settings >`, `smtrat::NRAILModule< Settings >`, `smtrat::NewCADModule< Settings >`, `smtrat::LVEModule< Settings >`, `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, `smtrat::LRAModule< LRASettings1 >`, `smtrat::IntEqModule< Settings >`, `smtrat::IntBlastModule< Settings >`, `smtrat::IncWidthModule< Settings >`, `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, `smtrat::ICPModule< ICPSettings1 >`, `smtrat::GBModule< Settings >`, `smtrat::FouMoModule< Settings >`, `smtrat::EQPreprocessingModule< Settings >`, `smtrat::EQModule< Settings >`, `smtrat::CurryModule< Settings >`, `smtrat::CubeLIAModule< Settings >`, `smtrat::CSplitModule< Settings >`, `smtrat::BVModule< Settings >`, and `smtrat::BEModule< Settings >`.

**0.15.333.4.76 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.333.4.77 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.333.4.78 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.333.4.79 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.333.4.80 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.333.4.81 `pReceivedFormula()`** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.333.4.82 print()** void smtrat::Module::print ( const std::string & \_initiation = "\*\*\*" ) const [inherited]  
Prints everything relevant of the solver.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.333.4.83 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & \_out = std::cout ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.333.4.84 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & \_initiation = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.333.4.85 printModel()** void smtrat::Module::printModel ( std::ostream & \_out = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.333.4.86 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & \_initiation = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.333.4.87 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & \_initiation = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

## Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.333.4.88 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

## Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

## Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.333.4.89 `processAnswer()`** `template<typename Settings > void smtrat::NewCoveringModule< Settings >::processAnswer ( )`

Processes the current answer, i.e.

computes a model or adds the infeasible subset.

**0.15.333.4.90 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.333.4.91 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.333.4.92 `receivedVariable()`** `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.333.4.93 `remove()`** `void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

## Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.333.4.94 removeConstraintsSAT()** template<typename Settings >  
void smtrat::NewCoveringModule< Settings >::removeConstraintsSAT ( )  
Removes the given constraints from the backend.

**Note**

This also removes all stored information about the removed constraints

**0.15.333.4.95 removeConstraintsUNSAT()** template<typename Settings >  
bool smtrat::NewCoveringModule< Settings >::removeConstraintsUNSAT ( )  
Removes the given constraints from the backend Also removes all stored information about the removed constraints.

**Returns**

True: If the model is still unsat after removing the constraints False: If the model might needs recomputation for the higher levels

**0.15.333.4.96 removeCore()** template<typename Settings >  
void smtrat::NewCoveringModule< Settings >::removeCore (   
    ModuleInput::const\_iterator \_subformula ) [virtual]  
Removes the subformula of the received formula at the given position to the considered ones of this module.  
Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

**Parameters**

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.333.4.97 removeOrigin()** std::pair<ModuleInput::iterator,bool> smtrat::Module::remove←  
Origin (   
    ModuleInput::iterator \_formula,  
    const FormulaT & \_origin ) [inline], [protected], [inherited]

**0.15.333.4.98 removeOrigins()** std::pair<ModuleInput::iterator,bool> smtrat::Module::remove←  
Origins (   
    ModuleInput::iterator \_formula,  
    const std::shared\_ptr< std::vector< FormulaT >> & \_origins ) [inline], [protected],  
[inherited]

**0.15.333.4.99 rPassedFormula()** const ModuleInput& smtrat::Module::rPassedFormula ( ) const  
[inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.333.4.100 rReceivedFormula()** const ModuleInput& smtrat::Module::rReceivedFormula ( ) const  
[inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.333.4.101 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.333.4.102 runBackends() [2/2]** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.333.4.103 setId()** `void smtrat::Module::setId (`

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

**Parameters**

|                 |                                    |
|-----------------|------------------------------------|
| <code>id</code> | The id to set this module's id to. |
|-----------------|------------------------------------|

**0.15.333.4.104 setThreadPriority()** `void smtrat::Module::setThreadPriority (`

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.333.4.105 solverState()** `Answer smtrat::Module::solverState () const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.333.4.106 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                           |
|---------------------------------|-----------------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol <code>!=</code> . |
|---------------------------------|-----------------------------------------------------------|

**0.15.333.4.107 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.333.4.108 `updateAllModels()`** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.333.4.109 `updateLemmas()`** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.333.4.110 `updateModel()`** `template<typename Settings >`

`void smtrat::NewCoveringModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.333.4.111 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.333.5 Field Documentation****0.15.333.5.1 `mAllBackends`** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.333.5.2 mAllModels** `std::vector<Model>` `smtrat::Module::mAllModels` [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.333.5.3 mBackendsFoundAnswer** `std::atomic_bool*` `smtrat::Module::mBackendsFoundAnswer` [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.333.5.4 mConstraintsToInform** `carl::FastSet<FormulaT>` `smtrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.333.5.5 mFinalCheck** `bool` `smtrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.333.5.6 mFirstPosInLastBranches** `std::size_t` `smtrat::Module::mFirstPosInLastBranches` = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.333.5.7 mFirstSubformulaToPass** `ModuleInput::iterator` `smtrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.333.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator` `smtrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.333.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.333.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.333.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.333.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.333.5.13 mLastBranches** std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.333.5.14 mLemmas** std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.333.5.15 mModel** Model smtrat::Module::mModel [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.333.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.333.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.333.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.333.5.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.333.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.333.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.333.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.333.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.333.5.24 mUsedBackends** std::vector<[Module](#)\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.333.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.334 smtrat::NNFPreparation Struct Reference

```
#include <NNFRewriter.h>
```

### Public Member Functions

- [FormulaT rewrite\\_implication](#) (const [FormulaT](#) &, [FormulaT](#) &&premise, [FormulaT](#) &&conclusion)
- [FormulaT rewrite\\_ite](#) (const [FormulaT](#) &, [FormulaT](#) &&condition, [FormulaT](#) &&trueCase, [FormulaT](#) &&falseCase)
- [FormulaT rewrite\\_iff](#) (const [FormulaT](#) &, [FormulaSetT](#) &&subformulas)
- [FormulaT rewrite\\_xor](#) (const [FormulaT](#) &, [FormulasMultiT](#) &&subformulas)

### 0.15.334.1 Member Function Documentation

**0.15.334.1.1 rewrite\_iff()** [FormulaT](#) smtrat::NNFPreparation::rewrite\_iff ( const [FormulaT](#) &, [FormulaSetT](#) && subformulas ) [inline]

**0.15.334.1.2 rewrite\_implication()** [FormulaT](#) smtrat::NNFPreparation::rewrite\_implication ( const [FormulaT](#) &, [FormulaT](#) && premise, [FormulaT](#) && conclusion ) [inline]

**0.15.334.1.3 rewrite\_ite()** [FormulaT](#) smtrat::NNFPreparation::rewrite\_ite ( const [FormulaT](#) &, [FormulaT](#) && condition, [FormulaT](#) && trueCase, [FormulaT](#) && falseCase ) [inline]

**0.15.334.1.4 rewrite\_xor()** [FormulaT](#) smtrat::NNFPreparation::rewrite\_xor ( const [FormulaT](#) &, [FormulasMultiT](#) && subformulas ) [inline]

## 0.15.335 smtrat::NNFRewriter Class Reference

```
#include <NNFRewriter.h>
```

### Data Structures

- struct [remove\\_xor\\_first\\_arg](#)

## Public Member Functions

- `NNFRewriter ()`
- `void enter_not (const FormulaT &)`
- `FormulaT rewrite_not (const FormulaT &formula, FormulaT &&subformula)`
- `FormulaT rewrite_leaf (const FormulaT &formula)`
- `FormulaT rewrite_ueq (const FormulaT &formula)`
- `FormulaT rewrite_quantified (const FormulaT &formula, FormulaT &&subformula)`
- `FormulaT rewrite_or (const FormulaT &formula, FormulaSetT &&subformulas)`
- `FormulaT rewrite_and (const FormulaT &formula, FormulaSetT &&subformulas)`
- `void enter_xor (const FormulaT &formula)`
- `FormulaT rewrite_xor (const FormulaT &formula, FormulasMultiT &&subformulas)`

### 0.15.335.1 Constructor & Destructor Documentation

**0.15.335.1.1 `NNFRewriter()`** smtrat::NNFRewriter::NNFRewriter ( ) [inline]

### 0.15.335.2 Member Function Documentation

**0.15.335.2.1 `enter_not()`** void smtrat::NNFRewriter::enter\_not ( const FormulaT & ) [inline]

**0.15.335.2.2 `enter_xor()`** void smtrat::NNFRewriter::enter\_xor ( const FormulaT & formula ) [inline]

**0.15.335.2.3 `rewrite_and()`** FormulaT smtrat::NNFRewriter::rewrite\_and ( const FormulaT & formula, FormulaSetT && subformulas ) [inline]

**0.15.335.2.4 `rewrite_leaf()`** FormulaT smtrat::NNFRewriter::rewrite\_leaf ( const FormulaT & formula ) [inline]

**0.15.335.2.5 `rewrite_not()`** FormulaT smtrat::NNFRewriter::rewrite\_not ( const FormulaT & formula, FormulaT && subformula ) [inline]

**0.15.335.2.6 `rewrite_or()`** FormulaT smtrat::NNFRewriter::rewrite\_or ( const FormulaT & formula, FormulaSetT && subformulas ) [inline]

**0.15.335.2.7 `rewrite_quantified()`** FormulaT smtrat::NNFRewriter::rewrite\_quantified ( const FormulaT & formula, FormulaT && subformula ) [inline]

---

**0.15.335.2.8 `rewrite_ueq()`** `FormulaT smtrat::NNFRewriter::rewrite_ueq (`  
`const FormulaT & formula ) [inline]`

**0.15.335.2.9 `rewrite_xor()`** `FormulaT smtrat::NNFRewriter::rewrite_xor (`  
`const FormulaT & formula,`  
`FormulasMultiT && subformulas ) [inline]`

## 0.15.336 `benchmax::ssh::Node` Struct Reference

Specification of a computation node for the SSH backend.

```
#include <Node.h>
```

### Data Fields

- `std::string hostname`  
*Hostname to connect to.*
- `std::string username`  
*Username.*
- `std::string password`  
*Password (only used if public key authentication fails).*
- `int port`  
*Port (default is 22)*
- `unsigned long cores`  
*Number of cores we use per connection (default is 1)*
- `std::size_t connections`  
*Number of concurrent connections (default is 1)*

### 0.15.336.1 Detailed Description

Specification of a computation node for the SSH backend.

### 0.15.336.2 Field Documentation

**0.15.336.2.1 `connections`** `std::size_t benchmax::ssh::Node::connections`  
Number of concurrent connections (default is 1)

**0.15.336.2.2 `cores`** `unsigned long benchmax::ssh::Node::cores`  
Number of cores we use per connection (default is 1)

**0.15.336.2.3 `hostname`** `std::string benchmax::ssh::Node::hostname`  
Hostname to connect to.

**0.15.336.2.4 `password`** `std::string benchmax::ssh::Node::password`  
Password (only used if public key authentication fails).

**0.15.336.2.5 `port`** `int benchmax::ssh::Node::port`  
Port (default is 22)

**0.15.336.2.6 `username`** std::string benchmax::ssh::Node::username  
Username.

## 0.15.337 `delta::Node` Struct Reference

This class represents a node in a SMTLIB file.

```
#include <Node.h>
```

### Public Member Functions

- `Node ()`  
*Create an empty node.*
- `Node (const std::string &name, bool brackets=true)`  
*Create a node with a name.*
- `Node (const std::tuple< std::string, bool > &data)`  
*Create a node with a name.*
- `Node (const std::tuple< std::vector< Node >, bool > &data)`  
*Create a node with children.*
- `Node (const std::tuple< std::string, std::vector< Node >, bool > &data)`  
*Create a node with a name and children.*
- `Node (const std::string &name, const std::initializer_list< Node > &children, bool brackets=true)`
- `void recalculateSize ()`
- `std::size_t complexity () const`  
*Calculates the number of nodes.*
- `bool immutable () const`  
*Checks if this node is immutable.*
- `std::string repr (bool longRepr=false) const`  
*Returns a string representation of this node.*
- `Node clone (const Node *from, const Node *to) const`  
*Clone this node recursively.*
- `Node clone (const std::string &from, const Node *to) const`  
*Clone this node recursively.*
- `void collectNames (std::set< std::string > &names) const`
- `void eliminateDefineFuns ()`

### Data Fields

- `std::string name`  
*Name of the node.*
- `std::vector< Node > children`  
*Children of the node.*
- `bool brackets`  
*Flag if the node was contained in brackets.*
- `std::size_t size`

### 0.15.337.1 Detailed Description

This class represents a node in a SMTLIB file.

Every syntactical constructor is represented by a node. A node may have a name and may have several children. In a file, this corresponds to (`<name> <child> <child> ...`).

Note that these nodes are constructed by a parser that has no semantic knowledge of the smtlib format. Hence, the data representable by these nodes is only a very rough overapproximation of the actual smtlib format.

### 0.15.337.2 Constructor & Destructor Documentation

**0.15.337.2.1 Node() [1/6]** `delta::Node::Node ()` [inline], [explicit]  
Create an empty node.

**0.15.337.2.2 Node() [2/6]** `delta::Node::Node (`  
    `const std::string & name,`  
    `bool brackets = true )` [inline], [explicit]

Create a node with a name.

#### Parameters

|                       |                                  |
|-----------------------|----------------------------------|
| <code>name</code>     | Name of the node.                |
| <code>brackets</code> | If not is contained in brackets. |

**0.15.337.2.3 Node() [3/6]** `delta::Node::Node (`  
    `const std::tuple< std::string, bool > & data )` [inline], [explicit]  
Create a node with a name.

#### Parameters

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <code>data</code> | Tuple containing the name and the brackets flag. |
|-------------------|--------------------------------------------------|

**0.15.337.2.4 Node() [4/6]** `delta::Node::Node (`  
    `const std::tuple< std::vector< Node >, bool > & data )` [inline], [explicit]  
Create a node with children.

#### Parameters

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <code>data</code> | Tuple containing the children and the brackets flag. |
|-------------------|------------------------------------------------------|

**0.15.337.2.5 Node() [5/6]** `delta::Node::Node (`  
    `const std::tuple< std::string, std::vector< Node >, bool > & data )` [inline],  
[explicit]  
Create a node with a name and children.

#### Parameters

|                   |                                                                |
|-------------------|----------------------------------------------------------------|
| <code>data</code> | Tuple containing the name, the children and the brackets flag. |
|-------------------|----------------------------------------------------------------|

**0.15.337.2.6 Node() [6/6]** `delta::Node::Node (`  
    `const std::string & name,`  
    `const std::initializer_list< Node > & children,`  
    `bool brackets = true )` [inline], [explicit]

### 0.15.337.3 Member Function Documentation

**0.15.337.3.1 clone()** [1/2] `Node delta::Node::clone (`  
    `const Node * from,`  
    `const Node * to ) const [inline]`

Clone this node recursively.

If the node `from` is encountered, replace it with `to`. If `to` is a `nullptr`, remove it instead.

#### Parameters

|                   |                                    |
|-------------------|------------------------------------|
| <code>from</code> | <code>Node</code> to replace.      |
| <code>to</code>   | <code>Node</code> to replace with. |

#### Returns

Cloned node.

**0.15.337.3.2 clone()** [2/2] `Node delta::Node::clone (`  
    `const std::string & from,`  
    `const Node * to ) const [inline]`

Clone this node recursively.

If a node with name `from` without children is encountered, replace it with `to`.

#### Parameters

|                   |                                    |
|-------------------|------------------------------------|
| <code>from</code> | <code>Node</code> name to replace. |
| <code>to</code>   | <code>Node</code> to replace with. |

#### Returns

Cloned node.

**0.15.337.3.3 collectNames()** `void delta::Node::collectNames (`  
    `std::set< std::string > & names ) const [inline]`

**0.15.337.3.4 complexity()** `std::size_t delta::Node::complexity ( ) const [inline]`  
Calculates the number of nodes.

#### Returns

Number of nodes.

**0.15.337.3.5 eliminateDefineFuns()** `void delta::Node::eliminateDefineFuns ( ) [inline]`

**0.15.337.3.6 immutable()** `bool delta::Node::immutable ( ) const [inline]`

Checks if this node is immutable.

This method implements simple checks to prevent unnecessary solver errors if essential parts of the smtlib file are removed, for example the `set-logic` statement.

**Returns**

If node may be removed.

**0.15.337.3.7 recalculateSize()** void delta::Node::recalculateSize ( ) [inline]**0.15.337.3.8 repr()** std::string delta::Node::repr ( bool longRepr = false ) const [inline]

Returns a string representation of this node.

**Returns**

[String](#) of the node.

**0.15.337.4 Field Documentation****0.15.337.4.1 brackets** bool delta::Node::brackets

Flag if the node was contained in brackets.

**0.15.337.4.2 children** std::vector<[Node](#)> delta::Node::children

Children of the node.

**0.15.337.4.3 name** std::string delta::Node::name

Name of the node.

**0.15.337.4.4 size** std::size\_t delta::Node::size**0.15.338 delta::NodePrinter< pretty > Struct Template Reference**

```
#include <Node.h>
```

**Public Member Functions**

- [NodePrinter](#) (const [Node](#) &n)
- void [print](#) (std::ostream &os, const [Node](#) &n) const
- void [pretty\\_print](#) (std::ostream &os, const [Node](#) &n, std::string indent="") const

**Data Fields**

- const [Node](#) & [node](#)

**0.15.338.1 Constructor & Destructor Documentation****0.15.338.1.1 NodePrinter()** template<bool pretty>  
delta::NodePrinter< pretty >::NodePrinter ( const [Node](#) & n ) [inline]**0.15.338.2 Member Function Documentation**

```
0.15.338.2.1 pretty_print() template<bool pretty>
void delta::NodePrinter< pretty >::pretty_print (
 std::ostream & os,
 const Node & n,
 std::string indent = "") const [inline]
```

```
0.15.338.2.2 print() template<bool pretty>
void delta::NodePrinter< pretty >::print (
 std::ostream & os,
 const Node & n) const [inline]
```

### 0.15.338.3 Field Documentation

```
0.15.338.3.1 node template<bool pretty>
const Node& delta::NodePrinter< pretty >::node
```

## 0.15.339 smtrat::mcsat::onecellcad::recursive::NoHeuristic Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr int **heuristic** = 0

### 0.15.339.1 Field Documentation

```
0.15.339.1.1 heuristic constexpr int smtrat::mcsat::onecellcad::recursive::NoHeuristic::heuristic
= 0 [static], [constexpr]
```

## 0.15.340 smtrat::NRAILModule< Settings > Class Template Reference

```
#include <NRAILModule.h>
```

### Public Types

- typedef **Settings** **SettingsType**
- enum class **LemmaType** : unsigned { **NORMAL** = 0 , **PERMANENT** = 1 }

### Public Member Functions

- std::string **moduleName** () const
- **NRAILModule** (const **ModuleInput** \***\_formula**, **Conditionals** &**\_conditionals**, **Manager** \***\_manager**=nullptr)
- **~NRAILModule** ()
- bool **informCore** (const **FormulaT** &**\_constraint**)  
*Informs the module about the given constraint.*
- void **init** ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- bool **addCore** (**ModuleInput**::const\_iterator **\_subformula**)  
*The module has to take the given sub-formula of the received formula into account.*
- void **removeCore** (**ModuleInput**::const\_iterator **\_subformula**)  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- void **updateModel** () const

- **Answer checkCore ()**

*Updates the current assignment into the model.*
- **bool inform (const FormulaT &\_constraint)**

*Checks the received formula for consistency.*
- **void deinform (const FormulaT &\_constraint)**

*Informs the module about the given constraint.*
- **bool add (ModuleInput::const\_iterator \_subformula)**

*The inverse of informing about a constraint.*
- **bool add (ModuleInput::const\_iterator \_subformula)**

*The module has to take the given sub-formula of the received formula into account.*
- **virtual Answer check (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)**

*Checks the received formula for consistency.*
- **virtual void remove (ModuleInput::const\_iterator \_subformula)**

*Removes everything related to the given sub-formula of the received formula.*
- **virtual void updateAllModels ()**

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- **virtual std::list< std::vector< carl::Variable > > getModelEqualities () const**

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- **unsigned currentlySatisfiedByBackend (const FormulaT &\_formula) const**
- **virtual unsigned currentlySatisfied (const FormulaT &) const**
- **bool receivedVariable (carl::Variable::Arg \_var) const**
- **Answer solverState () const**
- **std::size\_t id () const**
- **void setId (std::size\_t \_id)**

*Sets this modules unique ID to identify itself.*
- **thread\_priority threadPriority () const**
- **void setThreadPriority (thread\_priority \_threadPriority)**

*Sets the priority of this module to get a thread for running its check procedure.*
- **const ModuleInput \* pReceivedFormula () const**
- **const ModuleInput & rReceivedFormula () const**
- **const ModuleInput \* pPassedFormula () const**
- **const ModuleInput & rPassedFormula () const**
- **const Model & model () const**
- **const std::vector< Model > & allModels () const**
- **const std::vector< FormulaSetT > & infeasibleSubsets () const**
- **const std::vector< Module \* > & usedBackends () const**
- **const carl::FastSet< FormulaT > & constraintsToInform () const**
- **const carl::FastSet< FormulaT > & informedConstraints () const**
- **void addLemma (const FormulaT &\_lemma, const LemmaType &\_lt=LemmaType::NORMAL, const FormulaT &\_preferredFormula=FormulaT(carl::FormulaType::TRUE))**

*Stores a lemma being a valid formula.*
- **bool hasLemmas ()**

*Checks whether this module or any of its backends provides any lemmas.*
- **void clearLemmas ()**

*Deletes all yet found lemmas.*
- **const std::vector< Lemma > & lemmas () const**
- **ModuleInput::const\_iterator firstUncheckedReceivedSubformula () const**
- **ModuleInput::const\_iterator firstSubformulaToPass () const**
- **void receivedFormulaChecked ()**

*Notifies that the received formulas has been checked.*
- **const smrat::Conditionals & answerFound () const**
- **bool isPreprocessor () const**

- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- `void updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfeasibleSubsetForMinimality (std::vector<FormulaSetT>::const_iterator _insubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- `virtual std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- `void printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- `void printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- `void printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- `void printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- `void printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- `static void freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- `static size_t mNumOfBranchVarsToStore = 5`

*The number of different variables to consider for a probable infinite loop of branchings.*
- `static std::vector< Branching > mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`

*Stores the last branches in a cycle buffer.*
- `static size_t mFirstPosInLastBranches = 0`

*The beginning of the cyclic buffer storing the last branches.*
- `static std::vector< FormulaT > mOldSplittingVariables`

*Reusable splitting variables.*

### Protected Member Functions

- virtual void `deinformCore` (const `FormulaT` &\_constraint)
 

*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const
 

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const
 

*Clears the assignment, if any was found.*
- void `clearModels` () const
 

*Clears all assignments, if any was found.*
- void `cleanModel` () const
 

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin` ()
- `ModuleInput::iterator passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
 

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const
 

*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const `Model` & `backendsModel` () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel` () const

- void `getBackendsAllModels () const`  
*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`  
*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`
- std::vector< `FormulaT` > `merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &)` const  
*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin (const std::vector< FormulaT > &origins) const`
- bool `probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const`  
*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`  
*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &&_premise=std::vector< FormulaT >())`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())`
- void `splitUnequalConstraint (const FormulaT &)`  
*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel () const`
- void `addInformationRelevantFormula (const FormulaT &formula)`  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas ()`  
*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel (LemmaLevel level)`  
*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint (const Model &_modelA, const Model &_modelB)`  
*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- `Manager *const mpManager`  
*A reference to the manager.*
- `Model mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`  
*Stores all satisfying assignments.*

- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals `mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`  
*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- ModuleInput::iterator `mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- carl::FastSet< `FormulaT` > `mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- ModuleInput::const\_iterator `mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned `mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > `mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.340.1 Member Typedef Documentation

```
0.15.340.1.1 SettingsType template<typename Settings >
typedef Settings smtrat::NRAILModule< Settings >::SettingsType
```

### 0.15.340.2 Member Enumeration Documentation

```
0.15.340.2.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.340.3 Constructor & Destructor Documentation

**0.15.340.3.1 `NRAILModule()`** template<typename Settings >  
smtrat::NRAILModule< Settings >::NRAILModule ( const ModuleInput \* \_formula, Conditionals & \_conditionals, Manager \* \_manager = nullptr )

**0.15.340.3.2 `~NRAILModule()`** template<typename Settings >  
smtrat::NRAILModule< Settings >::~NRAILModule ( )

### 0.15.340.4 Member Function Documentation

**0.15.340.4.1 `add()`** bool smtrat::Module::add ( ModuleInput::const\_iterator \_subformula ) [inherited]

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.340.4.2 `addConstraintToInform()`** void smtrat::Module::addConstraintToInform ( const FormulaT & \_constraint ) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

#### Parameters

|             |                        |
|-------------|------------------------|
| _constraint | The constraint to add. |
|-------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.340.4.3 `addCore()`** template<typename Settings >  
bool smtrat::NRAILModule< Settings >::addCore (

ModuleInput::const\_iterator \_subformula ) [virtual]

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.340.4.4 addInformationRelevantFormula() void smtrat::Module::addInformationRelevantFormula (
 const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

```
0.15.340.4.5 addLemma() void smtrat::Module::addLemma (
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

```
0.15.340.4.6 addOrigin() void smtrat::Module::addOrigin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

```
0.15.340.4.7 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>_subformula</i> | The sub-formula of the received formula to copy. |
|--------------------|--------------------------------------------------|

```
0.15.340.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat->
```

---

```
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
Adds the given formula to the passed formula with no origin.
Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will
be removed.
```

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.340.4.9 `addSubformulaToPassedFormula()` [2/3]** `std::pair<ModuleInput::iterator,bool> smrat->`

```
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
Adds the given formula to the passed formula.
```

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.340.4.10 `addSubformulaToPassedFormula()` [3/3]** `std::pair<ModuleInput::iterator,bool> smrat->`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
Adds the given formula to the passed formula.
```

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.340.4.11 allModels()** const std::vector<[Model](#)>& smtrat::Module::allModels () const [inline], [inherited]

**Returns**

All satisfying assignments, if existent.

**0.15.340.4.12 anAnswerFound()** bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.340.4.13 answerFound()** const [smtrat::Conditionals](#)& smtrat::Module::answerFound () const [inline], [inherited]

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.340.4.14 backendsModel()** const [Model](#) & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.340.4.15 branchAt() [1/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const [Rational](#) & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< [FormulaT](#) > & \_premise = std::vector<[FormulaT](#)>() ) [inline], [protected], [inherited]

**0.15.340.4.16 branchAt() [2/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const [Rational](#) & \_value, std::vector< [FormulaT](#) > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.340.4.17 branchAt() [3/4]** bool smtrat::Module::branchAt ( const [Poly](#) & \_polynomial, bool \_integral, const [Rational](#) & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< [FormulaT](#) > & \_premise = std::vector<[FormulaT](#)>() ) [inline], [protected], [inherited]

---

**0.15.340.4.18 branchAt() [4/4]** `bool smtrat::Module::branchAt (`

```

const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]

```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

**0.15.340.4.19 check()** [Answer](#) `smtrat::Module::check (`

```

bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]

```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.340.4.20 checkCore()** `template<typename Settings >`

```
Answer smtrat::NRAILModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.340.4.21 `checkInfSubsetForMinimality()`** `void smrat::Module::checkInfSubsetForMinimality ( std::vector< FormulaSetT >::const_iterator _infssubset,`  
`const std::string & _filename = "smaller_muses",`  
`unsigned _maxSizeDifference = 1 ) const [inherited]`

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.340.4.22 `checkModel()`** `unsigned smrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.340.4.23 `cleanModel()`** `void smrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.340.4.24 `clearLemmas()`** `void smrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.340.4.25 `clearModel()`** `void smrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.340.4.26 `clearModels()`** `void smrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.340.4.27 `clearPassedFormula()`** `void smrat::Module::clearPassedFormula ( ) [protected], [inherited]`

---

**0.15.340.4.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inherited]

**0.15.340.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.340.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.340.4.31 constraintsToInform()** const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

#### Returns

The constraints which the backends must be informed about.

**0.15.340.4.32 currentlySatisfied()** virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAbstractModule< Settings >](#), [smtrat::LRAbstractModule< LRASettingsICP >](#), and [smtrat::LRAbstractModule< LRASettings >](#)

**0.15.340.4.33 currentlySatisfiedByBackend()** unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & \_formula ) const [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.340.4.34 deinform()** void smtrat::Module::deinform ( const FormulaT & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

## Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.340.4.35 `deinformCore()`** `virtual void smtrat::Module::deinformCore (``const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

## Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.340.4.36 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (``const std::vector< FormulaT > & origins ) const [protected], [inherited]`

## Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

## Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.340.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (``ModuleInput::iterator _subformula,``bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

## Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

## Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.340.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.340.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smrat::Module::findBestOrigin (`  
`const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

**Returns**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.340.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.340.4.41 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.340.4.42 `freeSplittingVariable()`** `static void smrat::Module::freeSplittingVariable (`  
`const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.340.4.43 `generateTrivialInfeasibleSubset()`** `void smrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.340.4.44 `getBackendsAllModels()`** `void smrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.340.4.45 `getBackendsModel()`** `void smrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.340.4.46 `getInfeasibleSubsets()` [1/2]** `void smrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.340.4.47 getInfeasibleSubsets()** [2/2] std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets (

    const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.340.4.48 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.340.4.49 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.340.4.50 `getOrigins()` [1/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.340.4.51 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.340.4.52 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.340.4.53 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified( ) [virtual], [inherited]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.340.4.54 `hasLemmas()`** `bool smtrat::Module::hasLemmas( ) [inline], [inherited]`

Checks whether this module or any of its backends provides any lemmas.

**0.15.340.4.55 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.340.4.56 `id()`** `std::size_t smtrat::Module::id( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.340.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.340.4.58 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.340.4.59 informBackends()** `void smrat::Module::informBackends (`

```
 const FormulaT & _constraint) [inline], [protected], [inherited]
```

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.340.4.60 informCore()** `template<typename Settings >`

```
bool smrat::NRAILModule< Settings >::informCore (
```

  

```
 const FormulaT & _constraint) [virtual]
```

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.340.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informed→`

```
Constraints () const [inline], [inherited]
```

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.340.4.62 init()** `template<typename Settings >`

```
void smrat::NRAILModule< Settings >::init () [virtual]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not and will not be called more than once and only before the first runBackends call.

Reimplemented from [smrat::Module](#).

**0.15.340.4.63 is\_minimizing()** `bool smrat::Module::is_minimizing ( ) const [inline], [inherited]`**0.15.340.4.64 isLemmaLevel()** `bool smrat::Module::isLemmaLevel (`

```
 LemmaLevel level) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.340.4.65 `isPreprocessor()`** `bool smtrat::Module::isPreprocessor () const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.340.4.66 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas () const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.340.4.67 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.340.4.68 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.340.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                |                                |
|----------------|--------------------------------|
| <i>_modelA</i> | The first model to check for.  |
| <i>_modelB</i> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.340.4.70 `moduleName()`** template<typename Settings >  
std::string smrat::NRAILModule< Settings >::moduleName ( ) const [inline], [virtual]

**Returns**

The name of the given module type as name.

Reimplemented from [smrat::Module](#).

**0.15.340.4.71 `objective()`** carl::Variable smrat::Module::objective ( ) const [inline], [inherited]

**0.15.340.4.72 `originInReceivedFormula()`** bool smrat::Module::originInReceivedFormula (const FormulaT & \_origin ) const [protected], [inherited]

**0.15.340.4.73 `passedFormulaBegin()`** ModuleInput::iterator smrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.340.4.74 `passedFormulaEnd()`** ModuleInput::iterator smrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.340.4.75 `pPassedFormula()`** const ModuleInput\* smrat::Module::pPassedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.340.4.76 `pReceivedFormula()`** const ModuleInput\* smrat::Module::pReceivedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.340.4.77 `print()`** void smrat::Module::print (const std::string & \_initiation = "\*\*\*" ) const [inherited]

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.340.4.78 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.340.4.79 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.340.4.80 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.340.4.81 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.340.4.82 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.340.4.83 probablyLooping()** `bool smtrat::Module::probablyLooping (`

```
const typename Poly::PolyType & _branchingPolynomial,
const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

**Parameters**

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

**Returns**

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.340.4.84 receivedFormulaChecked()** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.340.4.85 receivedFormulasAsInfeasibleSubset()** `void smtrat::Module::receivedFormulasAsInfeasibleSubset (`

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

**0.15.340.4.86 receivedVariable()** `bool smtrat::Module::receivedVariable (`

```
carl::Variable::Arg _var) const [inline], [inherited]
```

**0.15.340.4.87 remove()** `void smtrat::Module::remove (`

```
ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.340.4.88 removeCore()** `template<typename Settings >`

```
void smtrat::NRailModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

**Parameters**

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.340.4.89 `removeOrigin()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.340.4.90 `removeOrigins()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (`

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
 [inherited]
```

**0.15.340.4.91 `rPassedFormula()`** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const`

`[inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.340.4.92 `rReceivedFormula()`** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const`

`[inline], [inherited]`

#### Returns

A reference to the formula passed to this module.

**0.15.340.4.93 `runBackends() [1/2]`** `virtual Answer smtrat::Module::runBackends ( ) [inline],`

`[protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.340.4.94 `runBackends() [2/2]`** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.340.4.95 setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]  
Sets this modules unique ID to identify itself.

Parameters

|                             |                                    |
|-----------------------------|------------------------------------|
| $\leftrightarrow$<br>$\_id$ | The id to set this module's id to. |
|-----------------------------|------------------------------------|

**0.15.340.4.96 setThreadPriority()** void smtrat::Module::setThreadPriority ( thread\_priority \_threadPriority ) [inline], [inherited]  
Sets the priority of this module to get a thread for running its check procedure.

Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| $\_threadPriority$ | The priority to set this module's thread priority to. |
|--------------------|-------------------------------------------------------|

**0.15.340.4.97 solverState()** Answer smtrat::Module::solverState ( ) const [inline], [inherited]

Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.340.4.98 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]  
Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

Parameters

|                       |                                                  |
|-----------------------|--------------------------------------------------|
| $\_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|-----------------------|--------------------------------------------------|

**0.15.340.4.99 threadPriority()** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]

Returns

The priority of this module to get a thread for running its check procedure.

**0.15.340.4.100 updateAllModels()** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]  
Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.340.4.101 updateLemmas()** void smtrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.340.4.102 updateModel()** template<typename Settings >  
void smtrat::NRAILModule< Settings >::updateModel () const [virtual]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.340.4.103 usedBackends()** const std::vector<[Module](#)\*>& smtrat::Module::usedBackends ()  
const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.340.5 Field Documentation

**0.15.340.5.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected],  
[inherited]

The backends of this module which have been used.

**0.15.340.5.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected],  
[inherited]

Stores all satisfying assignments.

**0.15.340.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer  
[protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.340.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform  
[protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.340.5.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.340.5.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.340.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smtrat::Module::mFirstSubformulaTo←  
Pass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.340.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.340.5.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.340.5.10 mFullCheck** `bool smtrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.340.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`  
Stores the infeasible subsets.

**0.15.340.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.340.5.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`  
Stores the last branches in a cycle buffer.

**0.15.340.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.340.5.15 mModel** `Model smtrat::Module::mModel [mutable], [protected], [inherited]`  
Stores the assignment of the current satisfiable result, if existent.

**0.15.340.5.16 mModelComputed** `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`  
True, if the model has already been computed.

**0.15.340.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.340.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.340.5.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables [static], [inherited]`  
Reusable splitting variables.

**0.15.340.5.20 mpManager** `Manager* const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.340.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter`  
[mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.340.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected],  
[inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.340.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheory->`  
Propagations [protected], [inherited]

**0.15.340.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends` [protected],  
[inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.340.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters`  
[protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.341 smtrat::parser::NumeralParser Struct Reference

Parses numerals: (0 | [1-9] [0-9]\*)  
`#include <Lexicon.h>`

### 0.15.341.1 Detailed Description

Parses numerals: (0 | [1-9] [0-9]\*)

## 0.15.342 smtrat::lra::Numeric Class Reference

`#include <Numeric.h>`

### Public Member Functions

- **Numeric ()**  
*Default constructor.*
- **Numeric (const Rational &)**  
*Constructing from a Rational.*
- **Numeric (int)**  
*Constructing from an integer.*
- **Numeric (unsigned int)**  
*Constructing from an unsigned integer.*
- **Numeric (long)**  
*Constructing from a long integer.*
- **Numeric (unsigned long)**  
*Constructing from an unsigned long integer.*

- `Numeric (const char *)`  
*Constructing from a char array.*
- `Numeric (const Numeric &)`  
*Copy constructor.*
- `~Numeric ()`
- `const Rational & content () const`
- `Rational & rContent ()`
- `Numeric & operator= (int)`  
*Cast from an integer.*
- `Numeric & operator= (unsigned int)`  
*Cast from an unsigned integer.*
- `Numeric & operator= (long)`  
*Cast from a long integer.*
- `Numeric & operator= (unsigned long)`  
*Cast from an unsigned long integer.*
- `Numeric & operator= (const char *)`  
*Cast from a char array.*
- `Numeric & operator= (const Numeric &)`  
*Cast from a char array.*
- `bool operator== (const Numeric &) const`  
*Compares whether this `Numeric` and the given one are equal.*
- `bool operator!= (const Numeric &) const`  
*Compares whether this `Numeric` and the given one are not equal.*
- `bool operator< (const Numeric &) const`  
*Compares whether this `Numeric` is less than the given one.*
- `bool operator<= (const Numeric &) const`  
*Compares whether this `Numeric` is less or equal than the given one.*
- `bool operator> (const Numeric &) const`  
*Compares whether this `Numeric` is greater than the given one.*
- `bool operator>= (const Numeric &) const`  
*Compares whether this `Numeric` is greater or equal than the given one.*
- `Numeric numer () const`
- `Numeric denom () const`
- `Numeric floor () const`
- `bool is_positive () const`  
*Checks whether this `Numeric` corresponds to a positive rational number.*
- `bool is_negative () const`
- `bool is_zero () const`
- `bool is_integer () const`

#### 0.15.342.1 Constructor & Destructor Documentation

**0.15.342.1.1 `Numeric()` [1/8]** smtrat::lra::Numeric::Numeric ( )  
Default constructor.

**0.15.342.1.2 `Numeric()` [2/8]** smtrat::lra::Numeric::Numeric (   
  `const Rational & _value` )  
Constructing from a Rational.

**Parameters**

|            |           |
|------------|-----------|
| <i>The</i> | Rational. |
|------------|-----------|

**0.15.342.1.3 Numeric() [3/8]** smtrat::lra::Numeric::Numeric ( int \_value )

Constructing from an integer.

**Parameters**

|               |              |
|---------------|--------------|
| <i>_value</i> | The integer. |
|---------------|--------------|

**0.15.342.1.4 Numeric() [4/8]** smtrat::lra::Numeric::Numeric ( unsigned int \_value )

Constructing from an unsigned integer.

**Parameters**

|               |                      |
|---------------|----------------------|
| <i>_value</i> | The unsigned integer |
|---------------|----------------------|

**0.15.342.1.5 Numeric() [5/8]** smtrat::lra::Numeric::Numeric ( long \_value )

Constructing from a long integer.

**Parameters**

|               |                            |
|---------------|----------------------------|
| <i>_value</i> | The unsigned long integer. |
|---------------|----------------------------|

**0.15.342.1.6 Numeric() [6/8]** smtrat::lra::Numeric::Numeric ( unsigned long \_value )

Constructing from an unsigned long integer.

**Parameters**

|               |                            |
|---------------|----------------------------|
| <i>_value</i> | The unsigned long integer. |
|---------------|----------------------------|

**0.15.342.1.7 Numeric() [7/8]** smtrat::lra::Numeric::Numeric ( const char \* \_value )

Constructing from a char array.

**Parameters**

|               |                 |
|---------------|-----------------|
| <i>_value</i> | The char array. |
|---------------|-----------------|

**0.15.342.1.8 Numeric() [8/8]** smtrat::lra::Numeric::Numeric (

const Numeric &amp; \_value )

Copy constructor.

## Parameters

|        |                      |
|--------|----------------------|
| _value | The Numeric to copy. |
|--------|----------------------|

**0.15.342.1.9 ~Numeric()** smtrat::lra::Numeric::~Numeric ( )**0.15.342.2 Member Function Documentation****0.15.342.2.1 content()** const Rational& smtrat::lra::Numeric::content ( ) const [inline]**0.15.342.2.2 denom()** Numeric smtrat::lra::Numeric::denom ( ) const

## Returns

The denominator of this Numeric.

**0.15.342.2.3 floor()** Numeric smtrat::lra::Numeric::floor ( ) const

## Returns

The next smaller integer to this Numeric.

**0.15.342.2.4 is\_integer()** bool smtrat::lra::Numeric::is\_integer ( ) const

## Returns

true, if this Numeric is integer; false, otherwise.

**0.15.342.2.5 is\_negative()** bool smtrat::lra::Numeric::is\_negative ( ) const

## Returns

true, if this Numeric corresponds to a positive rational number; false, otherwise.

**0.15.342.2.6 is\_positive()** bool smtrat::lra::Numeric::is\_positive ( ) const

Checks whether this Numeric corresponds to a positive rational number.

## Returns

True, if this Numeric corresponds to a positive rational number; False, otherwise.

**0.15.342.2.7 `is_zero()`** `bool smtrat::lra::Numeric::is_zero ( ) const`**Returns**

true, if this [Numeric](#) corresponds to zero; false, otherwise.

**0.15.342.2.8 `numer()`** `Numeric smtrat::lra::Numeric::numer ( ) const`**Returns**

The enumerator of this [Numeric](#).

**0.15.342.2.9 `operator"!="()`** `bool smtrat::lra::Numeric::operator!= ( const Numeric & _value ) const`

Compares whether this [Numeric](#) and the given one are not equal.

**Parameters**

|                     |                                              |
|---------------------|----------------------------------------------|
| <code>_value</code> | The <a href="#">Numeric</a> to compare with. |
|---------------------|----------------------------------------------|

**Returns**

True, if the two Numerics are not equal; False, otherwise.

**0.15.342.2.10 `operator<()`** `bool smtrat::lra::Numeric::operator< ( const Numeric & _value ) const`

Compares whether this [Numeric](#) is less than the given one.

**Parameters**

|                     |                                              |
|---------------------|----------------------------------------------|
| <code>_value</code> | The <a href="#">Numeric</a> to compare with. |
|---------------------|----------------------------------------------|

**Returns**

True, if this [Numeric](#) is less than the given one; False, otherwise.

**0.15.342.2.11 `operator<=()`** `bool smtrat::lra::Numeric::operator<= ( const Numeric & _value ) const`

Compares whether this [Numeric](#) is less or equal than the given one.

**Parameters**

|                     |                                              |
|---------------------|----------------------------------------------|
| <code>_value</code> | The <a href="#">Numeric</a> to compare with. |
|---------------------|----------------------------------------------|

**Returns**

True, if this [Numeric](#) is less or equal than the given one; False, otherwise.

**0.15.342.2.12 `operator=()` [1/6]** `Numeric & smtrat::lra::Numeric::operator= ( const char * _value )`

Cast from a char array.

**Parameters**

|                     |                 |
|---------------------|-----------------|
| <code>_value</code> | The char array. |
|---------------------|-----------------|

**Returns**

The corresponding [Numeric](#).

**0.15.342.2.13 operator=()** [2/6] `Numeric & smtrat::lra::Numeric::operator= ( const Numeric & _value )`

Cast from a char array.

**Parameters**

|                     |                 |
|---------------------|-----------------|
| <code>_value</code> | The char array. |
|---------------------|-----------------|

**Returns**

The corresponding [Numeric](#).

**0.15.342.2.14 operator=()** [3/6] `Numeric & smtrat::lra::Numeric::operator= ( int _value )`

Cast from an integer.

**Parameters**

|                     |              |
|---------------------|--------------|
| <code>_value</code> | The integer. |
|---------------------|--------------|

**Returns**

The corresponding [Numeric](#).

**0.15.342.2.15 operator=()** [4/6] `Numeric & smtrat::lra::Numeric::operator= ( long _value )`

Cast from a long integer.

**Parameters**

|                     |                   |
|---------------------|-------------------|
| <code>_value</code> | The long integer. |
|---------------------|-------------------|

**Returns**

The corresponding [Numeric](#).

**0.15.342.2.16 operator=()** [5/6] `Numeric & smtrat::lra::Numeric::operator= ( unsigned int _value )`

Cast from an unsigned integer.

**Parameters**

|                     |                       |
|---------------------|-----------------------|
| <code>_value</code> | The unsigned integer. |
|---------------------|-----------------------|

**Returns**

The corresponding [Numeric](#).

**0.15.342.2.17 operator=()** [6/6] `Numeric & smtrat::lra::Numeric::operator= (`  
`unsigned long _value )`

Cast from an unsigned long integer.

**Parameters**

|                     |                            |
|---------------------|----------------------------|
| <code>_value</code> | The unsigned long integer. |
|---------------------|----------------------------|

**Returns**

The corresponding [Numeric](#).

**0.15.342.2.18 operator==( )** `bool smtrat::lra::Numeric::operator== (`  
`const Numeric & _value ) const`

Compares whether this [Numeric](#) and the given one are equal.

**Parameters**

|                     |                                              |
|---------------------|----------------------------------------------|
| <code>_value</code> | The <a href="#">Numeric</a> to compare with. |
|---------------------|----------------------------------------------|

**Returns**

True, if the two Numerics are equal; False, otherwise.

**0.15.342.2.19 operator>()** `bool smtrat::lra::Numeric::operator> (`  
`const Numeric & _value ) const`

Compares whether this [Numeric](#) is greater than the given one.

**Parameters**

|                     |                                              |
|---------------------|----------------------------------------------|
| <code>_value</code> | The <a href="#">Numeric</a> to compare with. |
|---------------------|----------------------------------------------|

**Returns**

True, if this [Numeric](#) is greater than the given one; False, otherwise.

**0.15.342.2.20 operator>=( )** `bool smtrat::lra::Numeric::operator>= (`  
`const Numeric & _value ) const`

Compares whether this [Numeric](#) is greater or equal than the given one.

**Parameters**

|                     |                                              |
|---------------------|----------------------------------------------|
| <code>_value</code> | The <a href="#">Numeric</a> to compare with. |
|---------------------|----------------------------------------------|

**Returns**

True, if this [Numeric](#) is greater or equal than the given one; False, otherwise.

**0.15.342.2.21 `rContent()`** `Rational& smtrat::lra::Numeric::rContent ()` [inline]

## 0.15.343 smtrat::execution::Objective Struct Reference

#include <ExecutionState.h>

**Data Fields**

- [Poly function](#)
- bool [minimize](#)

### 0.15.343.1 Field Documentation

**0.15.343.1.1 `function Poly`** smtrat::execution::Objective::function

**0.15.343.1.2 `minimize bool`** smtrat::execution::Objective::minimize

## 0.15.344 Minisat::OccLists< Idx, Vec, Deleted > Class Template Reference

#include <SolverTypes.h>

**Public Member Functions**

- [OccLists](#) (const Deleted &d)
- void [init](#) (const Idx &idx)
- const Vec & [operator\[\]](#) (const Idx &idx) const
- Vec & [operator\[\]](#) (const Idx &idx)
- Vec & [lookup](#) (const Idx &idx)
- void [cleanAll](#) ()
- void [clean](#) (const Idx &idx)
- void [smudge](#) (const Idx &idx)
- void [clear](#) (bool free=true)

### 0.15.344.1 Constructor & Destructor Documentation

**0.15.344.1.1 `OccLists()`** template<class Idx , class Vec , class Deleted >  
Minisat::OccLists< Idx, Vec, Deleted >::OccLists (const Deleted & d ) [inline]

### 0.15.344.2 Member Function Documentation

**0.15.344.2.1 clean()** template<class *Idx* , class *Vec* , class *Deleted* >  
void *Minisat::OccLists*< *Idx*, *Vec*, *Deleted* >::clean (  
  const *Idx* & *idx* )

**0.15.344.2.2 cleanAll()** template<class *Idx* , class *Vec* , class *Deleted* >  
void *Minisat::OccLists*< *Idx*, *Vec*, *Deleted* >::cleanAll

**0.15.344.2.3 clear()** template<class *Idx* , class *Vec* , class *Deleted* >  
void *Minisat::OccLists*< *Idx*, *Vec*, *Deleted* >::clear (  
  bool *free* = true ) [inline]

**0.15.344.2.4 init()** template<class *Idx* , class *Vec* , class *Deleted* >  
void *Minisat::OccLists*< *Idx*, *Vec*, *Deleted* >::init (  
  const *Idx* & *idx* ) [inline]

**0.15.344.2.5 lookup()** template<class *Idx* , class *Vec* , class *Deleted* >  
*Vec*& *Minisat::OccLists*< *Idx*, *Vec*, *Deleted* >::lookup (  
  const *Idx* & *idx* ) [inline]

**0.15.344.2.6 operator[]() [1/2]** template<class *Idx* , class *Vec* , class *Deleted* >  
*Vec*& *Minisat::OccLists*< *Idx*, *Vec*, *Deleted* >::operator[] ()  
  const *Idx* & *idx* ) [inline]

**0.15.344.2.7 operator[]() [2/2]** template<class *Idx* , class *Vec* , class *Deleted* >  
const *Vec*& *Minisat::OccLists*< *Idx*, *Vec*, *Deleted* >::operator[] ()  
  const *Idx* & *idx* ) const [inline]

**0.15.344.2.8 smudge()** template<class *Idx* , class *Vec* , class *Deleted* >  
void *Minisat::OccLists*< *Idx*, *Vec*, *Deleted* >::smudge (  
  const *Idx* & *idx* ) [inline]

## 0.15.345 smrat::mcsat::onecellcad::OneCellCAD Class Reference

```
#include <utils.h>
```

### Public Member Functions

- **OneCellCAD** (const std::vector< carl::Variable > &*variableOrder*, const RealAlgebraicPoint< Rational > &*point*)
- std::map< carl::Variable, RAN > **prefixPointToStdMap** (const std::size\_t *componentCount*)

*Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.*
- std::vector< RAN > **isolateLastVariableRoots** (const std::size\_t *polyLevel*, const Poly &*poly*)

*src/lib/datastructures/mcsat/onecellcad/OneCellCAD.h Given a poly p(x1,...,xn), return all roots of the univariate poly p(a1,...,an-1, xn), where a1..an-1 are the first algebraic real components from 'point' (SORTED!).*
- bool **vanishesEarly** (const std::size\_t *polyLevel*, const Poly &*poly*)

*Check if an n-variate 'poly' p(x1,...,xn) with n>=1 already becomes the zero poly after plugging in p(a1..,an-1, xn), where a1..an-1 are the first n-1 algebraic real components from 'point'.*
- bool **isPointRootOfPoly** (const std::size\_t *polyLevel*, const Poly &*poly*)

- bool `isPointRootOfPoly` (const `TagPoly` &`poly`)
- std::optional< `Poly` > `coeffNonNull` (const `TagPoly` &`boundCandidate`)
- bool `isMainPointInsideCell` (const `CADCell` &`cell`)

## Data Fields

- const std::vector< carl::Variable > & `variableOrder`  
*Variables can also be indexed by level.*
- const `RealAlgebraicPoint`< `Rational` > & `point`

### 0.15.345.1 Constructor & Destructor Documentation

**0.15.345.1.1 `OneCellCAD()`** smtrat::mcsat::oncelcad::OneCellCAD::OneCellCAD ( const std::vector< carl::Variable > & `variableOrder`, const `RealAlgebraicPoint`< `Rational` > & `point` ) [inline]

### 0.15.345.2 Member Function Documentation

**0.15.345.2.1 `coeffNonNull()`** std::optional<`Poly`> smtrat::mcsat::oncelcad::OneCellCAD::coeffNonNull ( const `TagPoly` & `boundCandidate` ) [inline]

**0.15.345.2.2 `isMainPointInsideCell()`** bool smtrat::mcsat::oncelcad::OneCellCAD::isMainPointInsideCell ( const `CADCell` & `cell` ) [inline]

**0.15.345.2.3 `isolateLastVariableRoots()`** std::vector<`RAN`> smtrat::mcsat::oncelcad::OneCellCAD:::isolateLastVariableRoots ( const std::size\_t `polyLevel`, const `Poly` & `poly` ) [inline]  
src/lib/datastructures/mcsat/oncelcad/OneCellCAD.h Given a poly p(x1,..,xn), return all roots of the univariate poly p(a1,..,an-1, xn), where a1,..,an-1 are the first algebraic real components from 'point' (SORTED!).

**0.15.345.2.4 `isPointRootOfPoly() [1/2]`** bool smtrat::mcsat::oncelcad::OneCellCAD::isPointRootOfPoly ( const std::size\_t `polyLevel`, const `Poly` & `poly` ) [inline]

**0.15.345.2.5 `isPointRootOfPoly() [2/2]`** bool smtrat::mcsat::oncelcad::OneCellCAD::isPointRootOfPoly ( const `TagPoly` & `poly` ) [inline]

**0.15.345.2.6 `prefixPointToStdMap()`** std::map<carl::Variable, `RAN`> smtrat::mcsat::oncelcad:::OneCellCAD::prefixPointToStdMap ( const std::size\_t `componentCount` ) [inline]

Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.

```
0.15.345.2.7 vanishesEarly() bool smtrat::mcsat::onecellcad::OneCellCAD::vanishesEarly (
 const std::size_t polyLevel,
 const Poly & poly) [inline]
```

Check if an n-variate 'poly'  $p(x_1, \dots, x_n)$  with  $n \geq 1$  already becomes the zero poly after plugging in  $p(a_1, \dots, a_{n-1}, x_n)$ , where  $a_1, \dots, a_{n-1}$  are the first  $n-1$  algebraic real components from 'point'.

### 0.15.345.3 Field Documentation

**0.15.345.3.1 point** const RealAlgebraicPoint<Rational>& smtrat::mcsat::onecellcad::OneCellCAD::point

**0.15.345.3.2 variableOrder** const std::vector<carl::Variable>& smtrat::mcsat::onecellcad::OneCellCAD::variableOrder

Variables can also be indexed by level.

Polys with mathematical level 1 contain the variable in variableOrder[0]

## 0.15.346 benchmax::settings::OperationSettings Struct Reference

Operation settings.

```
#include <Settings.h>
```

### Data Fields

- std::string **backend**  
*Name of the backend to be used.*
- std::string **mode**  
*Name of the operation mode.*
- std::string **convert\_ods\_filename**  
*Name of the xml file to convert to an ods file.*
- std::string **convert\_ods\_filter**  
*Name of the xslt filter used for ods import.*
- bool **use\_temp**  
*Use temporary directory.*

### 0.15.346.1 Detailed Description

Operation settings.

### 0.15.346.2 Field Documentation

**0.15.346.2.1 backend** std::string benchmax::settings::OperationSettings::backend  
Name of the backend to be used.

**0.15.346.2.2 convert\_ods\_filename** std::string benchmax::settings::OperationSettings::convert\_ods\_filename  
Name of the xml file to convert to an ods file.

**0.15.346.2.3 convert\_ods\_filter** std::string benchmax::settings::OperationSettings::convert\_ods←  
\_filter  
Name of the xslt filter used for ods import.

**0.15.346.2.4 mode** std::string benchmax::settings::OperationSettings::mode  
Name of the operation mode.

**0.15.346.2.5 use\_temp** bool benchmax::settings::OperationSettings::use\_temp  
Use temporary directory.

## 0.15.347 smrat::Optimization< Solver > Class Template Reference

```
#include <Optimization.h>
```

### Public Member Functions

- [Optimization \(Solver &s\)](#)
- void [add\\_objective \(const Poly &objective, bool minimize=true\)](#)
- void [remove\\_objective \(const Poly &objective\)](#)
- void [reset \(\)](#)
- bool [active \(\) const](#)
- std::tuple< [Answer](#), [Model](#), [ObjectiveValues](#) > [compute \(bool full=true\)](#)

### 0.15.347.1 Constructor & Destructor Documentation

**0.15.347.1.1 Optimization()** template<typename Solver >  
smrat::Optimization< Solver >::Optimization ( Solver & s ) [inline]

### 0.15.347.2 Member Function Documentation

**0.15.347.2.1 active()** template<typename Solver >  
bool smrat::Optimization< Solver >::active () const [inline]

**0.15.347.2.2 add\_objective()** template<typename Solver >  
void smrat::Optimization< Solver >::add\_objective ( const Poly & objective,  
bool minimize = true ) [inline]

**0.15.347.2.3 compute()** template<typename Solver >  
std::tuple< [Answer](#), [Model](#), [ObjectiveValues](#) > smrat::Optimization< Solver >::compute ( bool full = true ) [inline]

**0.15.347.2.4 remove\_objective()** template<typename Solver >  
void smrat::Optimization< Solver >::remove\_objective ( const Poly & objective ) [inline]

```
0.15.347.2.5 reset() template<typename Solver >
void smtrat::Optimization< Solver >::reset () [inline]
```

## 0.15.348 Minisat::Option Class Reference

```
#include <Options.h>
```

### Data Structures

- struct [OptionLt](#)

### Public Member Functions

- virtual [~Option \(\)](#)
- virtual void [help](#) (bool verbose=false)=0

### Protected Member Functions

- [Option](#) (const char \*name\_, const char \*desc\_, const char \*cate\_, const char \*type\_)

### Static Protected Member Functions

- static [vec< Option \\* > & getOptionList \(\)](#)
- static const char \*& [getUsageString \(\)](#)
- static const char \*& [getHelpPrefixString \(\)](#)

### Protected Attributes

- const char \* [name](#)
- const char \* [description](#)
- const char \* [category](#)
- const char \* [type\\_name](#)

### Friends

- void [printUsageAndExit](#) (int argc, char \*\*argv, bool verbose)
- void [setUsageHelp](#) (const char \*str)
- void [setHelpPrefixStr](#) (const char \*str)

## 0.15.348.1 Constructor & Destructor Documentation

```
0.15.348.1.1 Option() Minisat::Option::Option (
 const char * name_,
 const char * desc_,
 const char * cate_,
 const char * type_) [inline], [protected]
```

```
0.15.348.1.2 ~Option() virtual Minisat::Option::~Option () [inline], [virtual]
```

## 0.15.348.2 Member Function Documentation

```
0.15.348.2.1 getHelpPrefixString() static const char*& Minisat::Option::getHelpPrefixString ()
[inline], [static], [protected]
```

**0.15.348.2.2 `getOptionList()`** static `vec<Option*>&` Minisat::Option::getOptionList () [inline], [static], [protected]

**0.15.348.2.3 `getUsageString()`** static const `char*&` Minisat::Option::getUsageString () [inline], [static], [protected]

**0.15.348.2.4 `help()`** virtual void Minisat::Option::help ( bool verbose = false ) [pure virtual]

Implemented in [Minisat::BoolOption](#), [Minisat::StringOption](#), [Minisat::IntOption](#), and [Minisat::DoubleOption](#).

### 0.15.348.3 Friends And Related Function Documentation

**0.15.348.3.1 `printUsageAndExit()`** void printUsageAndExit ( int argc, char \*\* argv, bool verbose ) [friend]

**0.15.348.3.2 `setHelpPrefixStr()`** void setHelpPrefixStr ( const char \* str ) [friend]

**0.15.348.3.3 `setUsageHelp()`** void setUsageHelp ( const char \* str ) [friend]

### 0.15.348.4 Field Documentation

**0.15.348.4.1 `category`** const `char*` Minisat::Option::category [protected]

**0.15.348.4.2 `description`** const `char*` Minisat::Option::description [protected]

**0.15.348.4.3 `name`** const `char*` Minisat::Option::name [protected]

**0.15.348.4.4 `type_name`** const `char*` Minisat::Option::type\_name [protected]

## 0.15.349 Minisat::Option::OptionLt Struct Reference

#include <Options.h>

### Public Member Functions

- bool `operator()` (const Option \*x, const Option \*y)

### 0.15.349.1 Member Function Documentation

```
0.15.349.1.1 operator() bool Minisat::Option::OptionLt::operator() (
 const Option * x,
 const Option * y) [inline]
```

## 0.15.350 smtrat::datastructures::OrderPair< T1, T2, reversed > Struct Template Reference

```
#include <OrderPair.h>
```

### Public Member Functions

- `OrderPair (T1 pFirst=T1(), T2 pSecond=T2())`
- `bool operator< (const OrderPair< T1, T2, reversed > &rhs) const`

#### 0.15.350.1 Constructor & Destructor Documentation

```
0.15.350.1.1 OrderPair() template<typename T1 , typename T2 , bool reversed = false>
smtrat::datastructures::OrderPair< T1, T2, reversed >::OrderPair (
 T1 pFirst = T1(),
 T2 pSecond = T2()) [inline]
```

#### 0.15.350.2 Member Function Documentation

```
0.15.350.2.1 operator<() template<typename T1 , typename T2 , bool reversed = false>
bool smtrat::datastructures::OrderPair< T1, T2, reversed >::operator< (
 const OrderPair< T1, T2, reversed > & rhs) const [inline]
```

## 0.15.351 smtrat::datastructures::OrderPair< T1, T2, true > Struct Template Reference

```
#include <OrderPair.h>
```

### Public Member Functions

- `OrderPair (T1 pFirst=T1(), T2 pSecond=T2())`
- `bool operator< (const OrderPair< T1, T2, true > &rhs) const`

#### 0.15.351.1 Constructor & Destructor Documentation

```
0.15.351.1.1 OrderPair() template<typename T1 , typename T2 >
smtrat::datastructures::OrderPair< T1, T2, true >::OrderPair (
 T1 pFirst = T1(),
 T2 pSecond = T2()) [inline]
```

#### 0.15.351.2 Member Function Documentation

```
0.15.351.2.1 operator<() template<typename T1 , typename T2 >
bool smtrat::datastructures::OrderPair< T1, T2, true >::operator< (
 const OrderPair< T1, T2, true > & rhs) const [inline]
```

## 0.15.352 smtrat::cad::Origin Class Reference

This class represents one or more origins of some object.

```
#include <Origin.h>
```

### Data Structures

- struct [BaseType](#)

### Public Member Functions

- [Origin \(\)](#)
- [Origin \(const Origin &po\)](#)
- [Origin \(Origin &&po\)](#)
- [Origin \(std::size\\_t level, std::size\\_t id\)](#)
- [Origin \(const BaseType &bt\)](#)
- auto [begin \(\) const](#)
- auto [end \(\) const](#)
- bool [isActive \(\) const](#)
- void [deactivate \(std::size\\_t level, const carl::Bitset &rhs\)](#)
- void [deactivateEC \(std::size\\_t level, const carl::Bitset &rhs\)](#)
- void [activate \(std::size\\_t level, const carl::Bitset &rhs\)](#)
- void [activateEC \(std::size\\_t level, const carl::Bitset &rhs\)](#)
- [Origin & operator= \(const Origin &rhs\)](#)
- [Origin & operator= \(Origin &&rhs\)](#)
- bool [empty \(\) const](#)
- bool [operator== \(const Origin &rhs\) const](#)
- [Origin & operator+= \(const BaseType &rhs\)](#)  
*Adds the pair to the origins.*
- [Origin & operator-= \(const BaseType &rhs\)](#)  
*Removes the pair from the origins.*
- [Origin & erase \(std::size\\_t level, const carl::Bitset &rhs\)](#)
- [Origin operator| \(const Origin &rhs\) const](#)

### Friends

- std::ostream & [operator<< \(std::ostream &os, const Origin &po\)](#)

### 0.15.352.1 Detailed Description

This class represents one or more origins of some object.

An origin is a single id or a pair of ids (where a single id is represented as a pair of the same id).

As an object can have multiple origins, this class stores a list of such pairs. This class makes sure that the pairs are unique.

### 0.15.352.2 Constructor & Destructor Documentation

#### 0.15.352.2.1 [Origin\(\) \[1/5\]](#) smtrat::cad::Origin::Origin ( ) [inline]

#### 0.15.352.2.2 [Origin\(\) \[2/5\]](#) smtrat::cad::Origin::Origin ( const Origin & po ) [inline]

**0.15.352.2.3 `Origin()` [3/5]** smtrat::cad::Origin::Origin (   
   Origin && po ) [inline]

**0.15.352.2.4 `Origin()` [4/5]** smtrat::cad::Origin::Origin (   
   std::size\_t level,   
   std::size\_t id ) [inline], [explicit]

**0.15.352.2.5 `Origin()` [5/5]** smtrat::cad::Origin::Origin (   
   const BaseType & bt ) [inline], [explicit]

### 0.15.352.3 Member Function Documentation

**0.15.352.3.1 `activate()`** void smtrat::cad::Origin::activate (   
   std::size\_t level,   
   const carl::Bitset & rhs ) [inline]

**0.15.352.3.2 `activateEC()`** void smtrat::cad::Origin::activateEC (   
   std::size\_t level,   
   const carl::Bitset & rhs ) [inline]

**0.15.352.3.3 `begin()`** auto smtrat::cad::Origin::begin ( ) const [inline]

**0.15.352.3.4 `deactivate()`** void smtrat::cad::Origin::deactivate (   
   std::size\_t level,   
   const carl::Bitset & rhs ) [inline]

**0.15.352.3.5 `deactivateEC()`** void smtrat::cad::Origin::deactivateEC (   
   std::size\_t level,   
   const carl::Bitset & rhs ) [inline]

**0.15.352.3.6 `empty()`** bool smtrat::cad::Origin::empty ( ) const [inline]

**0.15.352.3.7 `end()`** auto smtrat::cad::Origin::end ( ) const [inline]

**0.15.352.3.8 `erase()`** Origin& smtrat::cad::Origin::erase (   
   std::size\_t level,   
   const carl::Bitset & rhs ) [inline]

**0.15.352.3.9 `isActive()`** bool smtrat::cad::Origin::isActive ( ) const [inline]

**0.15.352.3.10 operator+=()** `Origin& smtrat::cad::Origin::operator+= (`  
    `const BaseType & rhs ) [inline]`

Adds the pair to the origins.

**0.15.352.3.11 operator-=()** `Origin& smtrat::cad::Origin::operator-= (`  
    `const BaseType & rhs ) [inline]`

Removes the pair from the origins.

**0.15.352.3.12 operator=() [1/2]** `Origin& smtrat::cad::Origin::operator= (`  
    `const Origin & rhs ) [inline]`

**0.15.352.3.13 operator=() [2/2]** `Origin& smtrat::cad::Origin::operator= (`  
    `Origin && rhs ) [inline]`

**0.15.352.3.14 operator==()** `bool smtrat::cad::Origin::operator== (`  
    `const Origin & rhs ) const [inline]`

**0.15.352.3.15 operator"|"()** `Origin smtrat::cad::Origin::operator| (`  
    `const Origin & rhs ) const [inline]`

#### 0.15.352.4 Friends And Related Function Documentation

**0.15.352.4.1 operator<<** `std::ostream& operator<< (`  
    `std::ostream & os,`  
    `const Origin & po ) [friend]`

### 0.15.353 smtrat::cad::preprocessor::Origins Struct Reference

```
#include <CADPreprocessor.h>
```

#### Public Member Functions

- void `cleanOrigins (const FormulaT &f)`
- void `add (const FormulaT &f, const std::vector<FormulaT> &origin)`
- void `remove (const FormulaT &f)`
- const std::vector<FormulaT> & `get (const FormulaT &f) const`
- bool `resolve (std::set<FormulaT> &conflict) const`

#### Data Fields

- std::map<`FormulaT`, std::vector<std::vector<`FormulaT`>>> `mOrigins`

#### 0.15.353.1 Member Function Documentation

**0.15.353.1.1 add()** `void smtrat::cad::preprocessor::Origins::add (`  
    `const FormulaT & f,`  
    `const std::vector<FormulaT> & origin )`

```
0.15.353.1.2 cleanOrigins() void smtrat::cad::preprocessor::Origins::cleanOrigins (
 const FormulaT & f)
```

  

```
0.15.353.1.3 get() const std::vector< FormulaT > & smtrat::cad::preprocessor::Origins::get (
 const FormulaT & f) const
```

  

```
0.15.353.1.4 remove() void smtrat::cad::preprocessor::Origins::remove (
 const FormulaT & f)
```

  

```
0.15.353.1.5 resolve() bool smtrat::cad::preprocessor::Origins::resolve (
 std::set< FormulaT > & conflict) const
```

## 0.15.353.2 Field Documentation

**0.15.353.2.1 mOrigins** std::map<FormulaT, std::vector<std::vector<FormulaT>>> smtrat::cad::preprocessor::Origins::mOrigins

## 0.15.354 smtrat::OrPathShortener Struct Reference

A rewriter that checks every (X)OR that only contains ANDs and UEQs as children for equalities that hold for every alternative, and adds such equalities explicitly.

```
#include <OrPathShortener.h>
```

### Public Member Functions

- **OrPathShortener** (ModuleWrapper< EQModule< EQSettingsForPreprocessing >> &facts)
- **FormulaT rewrite\_or** (const FormulaT &formula, FormulaSetT &&subformulas)
- **FormulaT rewrite\_xor** (const FormulaT &formula, FormulasMultiT &&subformulas)
- **void enter\_default** (const FormulaT &formula)
- **FormulaT rewrite\_default** (const FormulaT &formula, const FormulaT &default\_result)

### 0.15.354.1 Detailed Description

A rewriter that checks every (X)OR that only contains ANDs and UEQs as children for equalities that hold for every alternative, and adds such equalities explicitly.

This should run after NNF transformation (but clearly before CNF transformation). This defeats eq\_diamond examples (and also more general cases where the paths are longer or irregular).

### 0.15.354.2 Constructor & Destructor Documentation

**0.15.354.2.1 OrPathShortener()** smtrat::OrPathShortener::OrPathShortener (
 ModuleWrapper< EQModule< EQSettingsForPreprocessing >> & facts ) [inline]

### 0.15.354.3 Member Function Documentation

**0.15.354.3.1 enter\_default()** void smtrat::OrPathShortener::enter\_default (
 const FormulaT & formula ) [inline]

**0.15.354.3.2 rewrite\_default()** `FormulaT smtrat::OrPathShortener::rewrite_default (`  
    `const FormulaT & formula,`  
    `const FormulaT & default_result ) [inline]`

**0.15.354.3.3 rewrite\_or()** `FormulaT smtrat::OrPathShortener::rewrite_or (`  
    `const FormulaT & formula,`  
    `FormulaSetT && subformulas ) [inline]`

**0.15.354.3.4 rewrite\_xor()** `FormulaT smtrat::OrPathShortener::rewrite_xor (`  
    `const FormulaT & formula,`  
    `FormulasMultiT && subformulas ) [inline]`

## 0.15.355 Minisat::OutOfMemoryException Class Reference

```
#include <XAlloc.h>
```

## 0.15.356 smtrat::parser::OutputWrapper Class Reference

```
#include <InstructionHandler.h>
```

### Public Member Functions

- `OutputWrapper (std::ostream &o, const std::string &prefix, const std::string &suffix)`
- `OutputWrapper (const OutputWrapper &&o)`
- `~OutputWrapper ()`
- template<typename T>  
    `OutputWrapper & operator<< (const T &t)`

### 0.15.356.1 Constructor & Destructor Documentation

**0.15.356.1.1 OutputWrapper() [1/2]** `smtrat::parser::OutputWrapper::OutputWrapper (`  
    `std::ostream & o,`  
    `const std::string & prefix,`  
    `const std::string & suffix ) [inline]`

**0.15.356.1.2 OutputWrapper() [2/2]** `smtrat::parser::OutputWrapper::OutputWrapper (`  
    `const OutputWrapper && o ) [inline]`

**0.15.356.1.3 ~OutputWrapper()** `smtrat::parser::OutputWrapper::~OutputWrapper ( ) [inline]`

### 0.15.356.2 Member Function Documentation

**0.15.356.2.1 operator<<()** `template<typename T >`  
`OutputWrapper& smtrat::parser::OutputWrapper::operator<< (`  
    `const T & t ) [inline]`

## 0.15.357 Minisat::Map< K, D, H, E >::Pair Struct Reference

```
#include <Map.h>
```

## Data Fields

- K [key](#)
- D [data](#)

### 0.15.357.1 Field Documentation

**0.15.357.1.1 data** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
D [Minisat::Map](#)< K, D, H, E >::Pair::data

**0.15.357.1.2 key** template<class K , class D , class H = Hash<K>, class E = Equal<K>>  
K [Minisat::Map](#)< K, D, H, E >::Pair::key

## 0.15.358 smtrat::pairhash< F, S, FirstHasher, SecondHasher > Struct Template Reference

Note that this may NOT be a specialization of std::hash for std::pair, this is not allowed by the standard as it does not involve a user-defined type.

```
#include <PairHash.h>
```

### Public Member Functions

- std::size\_t [operator\(\)](#) (const std::pair< F, S > &p) const noexcept

### 0.15.358.1 Detailed Description

```
template<typename F, typename S, typename FirstHasher = std::hash<F>, typename SecondHasher = std::hash<S>>
struct smtrat::pairhash< F, S, FirstHasher, SecondHasher >
```

Note that this may NOT be a specialization of std::hash for std::pair, this is not allowed by the standard as it does not involve a user-defined type.

### 0.15.358.2 Member Function Documentation

**0.15.358.2.1 operator()** template<typename F , typename S , typename FirstHasher = std::hash<F>, typename SecondHasher = std::hash<S>>
std::size\_t [smtrat::pairhash](#)< F, S, FirstHasher, SecondHasher >::operator() (
 const std::pair< F, S > & p ) const [inline], [noexcept]

## 0.15.359 smtrat::mcsat::ParallelExplanation< Backends > Struct Template Reference

```
#include <ParallelExplanation.h>
```

### Public Member Functions

- std::optional< [Explanation](#) > [operator\(\)](#) (const [mcsat::Bookkeeping](#) &data, carl::Variable var, const [FormulasT](#) &reason, bool force\_use\_core) const

### 0.15.359.1 Member Function Documentation

```
0.15.359.1.1 operator() template<typename... Backends>
std::optional<Explanation> smtrat::mcsat::ParallelExplanation< Backends >::operator() (
 const mcsat::Bookkeeping & data,
 carl::Variable var,
 const FormulasT & reason,
 bool force_use_core) const [inline]
```

## 0.15.360 delta::Parser Class Reference

This class parses a whole smtlib file into a hierarchy of [Node](#) objects.

```
#include <Parser.h>
```

### Public Member Functions

- [Parser \(\)](#)  
*Constructs the parsing rules.*
- [bool parseFile \(const std::string &filename, Node &node\)](#)  
*Parses a file into a node.*

### Static Public Member Functions

- [static bool parse \(const std::string &filename, Node &node\)](#)  
*Parses a file into a node.*

#### 0.15.360.1 Detailed Description

This class parses a whole smtlib file into a hierarchy of [Node](#) objects.

#### 0.15.360.2 Constructor & Destructor Documentation

**0.15.360.2.1 Parser()** `delta::Parser::Parser () [inline]`  
Constructs the parsing rules.

#### 0.15.360.3 Member Function Documentation

**0.15.360.3.1 parse()** `static bool delta::Parser::parse (`  
    `const std::string & filename,`  
    `Node & node ) [inline], [static]`  
Parses a file into a node.

##### Parameters

|                       |                             |
|-----------------------|-----------------------------|
| <code>filename</code> | Filename of the input file. |
|-----------------------|-----------------------------|

##### Returns

[Node](#) object produced by the parser.

**0.15.360.3.2 parseFile()** `bool delta::Parser::parseFile (`  
    `const std::string & filename,`  
    `Node & node ) [inline]`

Parses a file into a node.

**Parameters**

|                       |                             |
|-----------------------|-----------------------------|
| <code>filename</code> | Filename of the input file. |
|-----------------------|-----------------------------|

**Returns**

`Node` object produced by the parser.

**0.15.361 smtrat::parser::ParserSettings Struct Reference**

```
#include <ParserSettings.h>
```

**Data Fields**

- bool `read_dimacs`
- bool `read_opb`
- std::string `input_file`
- bool `disable_uf_flattening`
- bool `disable_theory`

**0.15.361.1 Field Documentation**

**0.15.361.1.1 disable\_theory** bool smtrat::parser::ParserSettings::disable\_theory

**0.15.361.1.2 disable\_uf\_flattening** bool smtrat::parser::ParserSettings::disable\_uf\_flattening

**0.15.361.1.3 input\_file** std::string smtrat::parser::ParserSettings::input\_file

**0.15.361.1.4 read\_dimacs** bool smtrat::parser::ParserSettings::read\_dimacs

**0.15.361.1.5 read\_opb** bool smtrat::parser::ParserSettings::read\_opb

**0.15.362 smtrat::parser::ParserState Struct Reference**

```
#include <ParserState.h>
```

**Data Structures**

- struct `ExpressionScope`
- struct `ScriptScope`

**Public Member Functions**

- `ParserState (InstructionHandler &ih)`
- `~ParserState ()`
- `void pushExpressionScope ()`
- `void popExpressionScope ()`
- `void pushScriptScope ()`
- `void popScriptScope ()`
- `std::size_t script_scope_size () const`

- void `reset()`
- void `errorMessage(const std::string &msg)`
- bool `isSymbolFree(const std::string &name, bool output=true)`
- template<typename Res , typename T >  
  bool `resolveSymbol(const std::string &name, const std::map< std::string, T > &map, Res &result) const`
- bool `resolveSymbol(const std::string &name, types::TermType &r) const`
- bool `resolveSymbol(const std::string &name, types::VariableType &r) const`
- void `registerFunction(const std::string &name, const FunctionInstantiator *fi)`
- void `registerFunction(const std::string &name, const IndexedFunctionInstantiator *fi)`
- void `registerFunction(const std::string &name, const UserFunctionInstantiator *fi)`

## Data Fields

- carl::Logic `logic`
- std::set< `types::VariableType` > `auxiliary_variables`
- std::map< `std::string`, `types::TermType` > `bindings`
- std::map< `std::string`, `types::TermType` > `constants`
- std::map< `std::string`, `types::VariableType` > `variables`
- std::map< `std::string`, carl::UninterpretedFunction > `declared_functions`
- std::map< `std::string`, const `FunctionInstantiator` \* > `defined_functions`
- std::map< `std::string`, const `IndexedFunctionInstantiator` \* > `defined_indexed_functions`
- std::map< `std::string`, const `UserFunctionInstantiator` \* > `defined_user_functions`
- `FormulasT global_formulas`
- std::vector< `smtrat::ModelVariable` > `artificialVariables`
- `InstructionHandler & handler`
- std::stack< `ExpressionScope` > `expressionScopes`
- std::stack< `ScriptScope` > `scriptScopes`

### 0.15.362.1 Constructor & Destructor Documentation

**0.15.362.1.1 ParserState()** smtrat::parser::ParserState::ParserState (   
  `InstructionHandler & ih` ) [inline]

**0.15.362.1.2 ~ParserState()** smtrat::parser::ParserState::~ParserState ( ) [inline]

### 0.15.362.2 Member Function Documentation

**0.15.362.2.1 errorMessage()** void smtrat::parser::ParserState::errorMessage (   
  `const std::string & msg` ) [inline]

**0.15.362.2.2 isSymbolFree()** bool smtrat::parser::ParserState::isSymbolFree (   
  `const std::string & name,`  
  `bool output = true` ) [inline]

**0.15.362.2.3 popExpressionScope()** void smtrat::parser::ParserState::popExpressionScope ( )  
[inline]

**0.15.362.2.4 `popScriptScope()`** void smtrat::parser::ParserState::popScriptScope ( ) [inline]

**0.15.362.2.5 `pushExpressionScope()`** void smtrat::parser::ParserState::pushExpressionScope ( ) [inline]

**0.15.362.2.6 `pushScriptScope()`** void smtrat::parser::ParserState::pushScriptScope ( ) [inline]

**0.15.362.2.7 `registerFunction()` [1/3]** void smtrat::parser::ParserState::registerFunction ( const std::string & name, const FunctionInstantiator \* fi ) [inline]

**0.15.362.2.8 `registerFunction()` [2/3]** void smtrat::parser::ParserState::registerFunction ( const std::string & name, const IndexedFunctionInstantiator \* fi ) [inline]

**0.15.362.2.9 `registerFunction()` [3/3]** void smtrat::parser::ParserState::registerFunction ( const std::string & name, const UserFunctionInstantiator \* fi ) [inline]

**0.15.362.2.10 `reset()`** void smtrat::parser::ParserState::reset ( ) [inline]

**0.15.362.2.11 `resolveSymbol()` [1/3]** template<typename Res , typename T > bool smtrat::parser::ParserState::resolveSymbol ( const std::string & name, const std::map< std::string, T > & map, Res & result ) const [inline]

**0.15.362.2.12 `resolveSymbol()` [2/3]** bool smtrat::parser::ParserState::resolveSymbol ( const std::string & name, types::TermType & r ) const [inline]

**0.15.362.2.13 `resolveSymbol()` [3/3]** bool smtrat::parser::ParserState::resolveSymbol ( const std::string & name, types::VariableType & r ) const [inline]

**0.15.362.2.14 `script_scope_size()`** std::size\_t smtrat::parser::ParserState::script\_scope\_size ( ) const [inline]

## 0.15.362.3 Field Documentation

**0.15.362.3.1 `artificialVariables`** std::vector<smtrat::ModelVariable> smtrat::parser::ParserState::artificialVariables

**0.15.362.3.2 auxiliary\_variables** std::set<[types::VariableType](#)> smtrat::parser::ParserState::auxiliary\_variables

**0.15.362.3.3 bindings** std::map<std::string, [types::TermType](#)> smtrat::parser::ParserState::bindings

**0.15.362.3.4 constants** std::map<std::string, [types::TermType](#)> smtrat::parser::ParserState::constants

**0.15.362.3.5 declared\_functions** std::map<std::string, carl::UninterpretedFunction> smtrat::parser::ParserState::declared\_functions

**0.15.362.3.6 defined\_functions** std::map<std::string, const [FunctionInstantiator](#)\*> smtrat::parser::ParserState::defined\_functions

**0.15.362.3.7 defined\_indexed\_functions** std::map<std::string, const [IndexedFunctionInstantiator](#)\*> smtrat::parser::ParserState::defined\_indexed\_functions

**0.15.362.3.8 defined\_user\_functions** std::map<std::string, const [UserFunctionInstantiator](#)\*> smtrat::parser::ParserState::defined\_user\_functions

**0.15.362.3.9 expressionScopes** std::stack<[ExpressionScope](#)> smtrat::parser::ParserState::expressionScopes

**0.15.362.3.10 global\_formulas** [FormulaST](#) smtrat::parser::ParserState::global\_formulas

**0.15.362.3.11 handler** [InstructionHandler](#)& smtrat::parser::ParserState::handler

**0.15.362.3.12 logic** carl::Logic smtrat::parser::ParserState::logic

**0.15.362.3.13 scriptScopes** std::stack<[ScriptScope](#)> smtrat::parser::ParserState::scriptScopes

**0.15.362.3.14 variables** std::map<std::string, [types::VariableType](#)> smtrat::parser::ParserState::variables

## 0.15.363 smtrat::PBGaussModule< Settings > Class Template Reference

#include <PBGaussModule.h>

## Public Types

- using `MatrixT` = Eigen::Matrix< Rational, Eigen::Dynamic, Eigen::Dynamic >
- using `VectorT` = Eigen::Matrix< Rational, Eigen::Dynamic, 1 >
- typedef `Settings SettingsType`
- enum class `LemmaType` : unsigned { `NORMAL` = 0 , `PERMANENT` = 1 }

## Public Member Functions

- std::string `moduleName () const`
- `PBGaussModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)`
- `~PBGaussModule ()`
- bool `informCore (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- void `init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- bool `addCore (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- void `removeCore (ModuleInput::const_iterator _subformula)`  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- void `updateModel () const`  
*Updates the current assignment into the model.*
- `Answer checkCore ()`  
*Checks the received formula for consistency.*
- `FormulaT gaussAlgorithm ()`
- `FormulaT reconstructEqSystem (const MatrixT &m, const VectorT &b, const carl::Variables &vars, const std::vector< carl::Relation > &rels)`
- `FormulaT reduce (const MatrixT &u, const VectorT &b, const carl::Variables vars)`
- `std::vector< Rational > lookForReductionRow (const MatrixT &uMatrix, const VectorT &ineqRow, long column)`
- bool `inform (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- void `deinform (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- bool `add (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- virtual `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
*Checks the received formula for consistency.*
- virtual void `remove (ModuleInput::const_iterator _subformula)`  
*Removes everything related to the given sub-formula of the received formula.*
- virtual void `updateAllModels ()`  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > `getModelEqualities () const`  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned `currentlySatisfiedByBackend (const FormulaT &_formula) const`
- virtual unsigned `currentlySatisfied (const FormulaT &) const`
- bool `receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- std::size\_t `id () const`
- void `setId (std::size_t _id)`  
*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`

- void `setThreadPriority (thread_priority _threadPriority)`  
*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput * pReceivedFormula () const`
- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const `std::vector< Model > & allModels () const`
- const `std::vector< FormulaSetT > & infeasibleSubsets () const`
- const `std::vector< Module * > & usedBackends () const`
- const `carl::FastSet< FormulaT > & constraintsToInform () const`
- const `carl::FastSet< FormulaT > & informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`  
*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`  
*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`  
*Deletes all yet found lemmas.*
- const `std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`  
*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`  
*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `isValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, FormulaT > `getReceivedFormulaSimplified ()`
- void `print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `conservativeResize` (`VectorT` &`v`, long `newSize`)  
*Convenience wrapper for Eigen::conservativeResizeLike with a zero vector.*
- static void `conservativeResize` (`MatrixT` &`m`, long `newRows`, long `newCols`)  
*Convenience wrapper for Eigen::conservativeResizeLike with a zero matrix.*
- static void `freeSplittingVariable` (const `FormulaT` &`_splittingVariable`)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< `Branching` > `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< `FormulaT` > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual void `deinformCore` (const `FormulaT` &`_constraint`)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` `_formula`, const `FormulaT` &`_origin`)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` `_formula`) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &`_formula`, `FormulasT` &`_origins`) const
- void `getOrigins` (const `FormulaT` &`_formula`, `FormulaSetT` &`_origins`) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` `_formula`, const `FormulaT` &`_origin`)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` `_formula`, const std::shared\_ptr< std::vector< `FormulaT` >> &`_origins`)
- void `informBackends` (const `FormulaT` &`_constraint`)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &`_constraint`)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` `_subformula`)  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &`_origin`) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &`_formula`)

- Adds the given formula to the passed formula with no origin.*

  - std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)

*Adds the given formula to the passed formula.*

  - std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)

*Adds the given formula to the passed formula.*

  - void [generateTrivialInfeasibleSubset](#) ()

*Stores the trivial infeasible subset being the set of received formulas.*

  - void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)

*Stores an infeasible subset consisting only of the given received formula.*

  - std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
  - void [getInfeasibleSubsets](#) ()

*Copies the infeasible subsets of the passed formula.*

  - std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const

*Get the infeasible subsets the given backend provides.*

  - const [Model & backendsModel](#) () const

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*

  - void [getBackendsModel](#) () const
  - void [getBackendsAllModels](#) () const

*Stores all models of a backend in the list of all models of this module.*

  - virtual [Answer runBackends](#) (bool \_final, bool \_full, carl::Variable \_objective)

*Runs the backend solvers on the passed formula.*

  - virtual [Answer runBackends](#) ()
  - virtual [ModuleInput::iterator eraseSubformulaFromPassedFormula](#) ([ModuleInput::iterator](#) \_subformula, bool \_ignoreOrigins=false)

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*

  - void [clearPassedFormula](#) ()
  - std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const

*Merges the two vectors of sets into the first one.*

  - size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
  - bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const [Rational](#) &← branchingValue) const

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*

  - bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*

  - bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &← \_premise=std::vector< [FormulaT](#) >())
  - bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
  - bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
  - void [splitUnequalConstraint](#) (const [FormulaT](#) &)

*Adds the following lemmas for the given constraint  $p \neq 0$ .*

  - unsigned [checkModel](#) () const
  - void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)

*Adds a formula to the InformationRelevantFormula.*

- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()
 

*Gets all InformationRelevantFormulas.*
- bool [isLemmaLevel](#) ([LemmaLevel](#) level)
 

*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool [modelsDisjoint](#) (const [Model](#) &\_modelA, const [Model](#) &\_modelB)
 

*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector< [FormulaSetT](#) > [mInfeasibleSubsets](#)

*Stores the infeasible subsets.*
- [Manager](#) \*const [mpManager](#)

*A reference to the manager.*
- [Model](#) [mModel](#)

*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< [Model](#) > [mAllModels](#)

*Stores all satisfying assignments.*
- bool [mModelComputed](#)

*True, if the model has already been computed.*
- bool [mFinalCheck](#)

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool [mFullCheck](#)

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable [mObjectiveVariable](#)

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< [TheoryPropagation](#) > [mTheoryPropagations](#)
- std::atomic< [Answer](#) > [mSolverState](#)

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* [mBackendsFoundAnswer](#)

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- [Conditionals](#) [mFoundAnswer](#)

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< [Module](#) \* > [mUsedBackends](#)

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< [Module](#) \* > [mAllBackends](#)

*The backends of this module which have been used.*
- std::vector< [Lemma](#) > [mLemmas](#)

*Stores the lemmas being valid formulas this module or its backends made.*
- [ModuleInput](#)::iterator [mFirstSubformulaToPass](#)

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< [FormulaT](#) > [mConstraintsToInform](#)

*Stores the constraints which the backends must be informed about.*
- carl::FastSet< [FormulaT](#) > [mInformedConstraints](#)

*Stores the position of the first constraint of which no backend has been informed about.*
- [ModuleInput](#)::const\_iterator [mFirstUncheckedReceivedSubformula](#)

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned [mSmallerMusesCheckCounter](#)

*Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > [mVariableCounters](#)

*Maps variables to the number of their occurrences.*

### 0.15.363.1 Member Typedef Documentation

**0.15.363.1.1 MatrixT** template<typename Settings >  
 using smtrat::PBGaussModule< Settings >::MatrixT = Eigen::Matrix< Rational, Eigen::Dynamic, Eigen::Dynamic>

**0.15.363.1.2 SettingsType** template<typename Settings >  
 typedef Settings smtrat::PBGaussModule< Settings >::SettingsType

**0.15.363.1.3 VectorT** template<typename Settings >  
 using smtrat::PBGaussModule< Settings >::VectorT = Eigen::Matrix< Rational, Eigen::Dynamic, 1>

### 0.15.363.2 Member Enumeration Documentation

**0.15.363.2.1 LemmaType** enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.363.3 Constructor & Destructor Documentation

**0.15.363.3.1 PBGaussModule()** template<typename Settings >  
 smtrat::PBGaussModule< Settings >::PBGaussModule ( const ModuleInput \* \_formula, Conditionals & \_conditionals, Manager \* \_manager = nullptr )

**0.15.363.3.2 ~PBGaussModule()** template<typename Settings >  
 smtrat::PBGaussModule< Settings >::~PBGaussModule ( )

### 0.15.363.4 Member Function Documentation

**0.15.363.4.1 add()** bool smtrat::Module::add ( ModuleInput::const\_iterator \_subformula ) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.363.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`

```
const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.363.4.3 addCore()** `template<typename Settings >`

```
bool smtrat::PBGaussModule< Settings >::addCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.363.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula`

```
(
```

```
const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.363.4.5 addLemma()** `void smtrat::Module::addLemma (`

```
const FormulaT & _lemma,
```

```
const LemmaType & _lt = LemmaType::NORMAL,
```

```
const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
```

```
[inherited]
```

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.363.4.6 addOrigin()** `void smtrat::Module::addOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.363.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.363.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.363.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator, bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.363.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>**

```
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.363.4.11 allModels() const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]****Returns**

All satisfying assignments, if existent.

**0.15.363.4.12 anAnswerFound() bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.363.4.13 answerFound() const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.363.4.14 backendsModel() const Model & smtrat::Module::backendsModel () const [protected], [inherited]**

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.363.4.15** **branchAt()** [1/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.363.4.16** **branchAt()** [2/4] `bool smrat::Module::branchAt (`

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.363.4.17** **branchAt()** [3/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.363.4.18** **branchAt()** [4/4] `bool smrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.363.4.19 check() Answer smrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::PModule](#).

```
0.15.363.4.20 checkCore() template<typename Settings >
Answer smrat::PBGaussModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

```
0.15.363.4.21 checkInfSubsetForMinimality() void smrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.363.4.22 checkModel() unsigned smrat::Module::checkModel () const [protected], [inherited]
```

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.363.4.23 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.363.4.24 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.363.4.25 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.363.4.26 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.363.4.27 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.363.4.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const **FormulaT** & \_formula, **FormulaSetT** & \_origins ) const [inherited]

**0.15.363.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const **FormulaT** & \_formula, **FormulasT** & \_origins ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                 |                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------|
| <b>_formula</b> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <b>_origins</b> | The set in which to store the origins.                                                               |

**0.15.363.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.363.4.31 conservativeResize() [1/2]** template<typename Settings >  
static void smtrat::PBGaussModule< Settings >::conservativeResize (

```
MatrixT & m,
long newRows,
long newCols) [inline], [static]
```

Convenience wrapper for Eigen::conservativeResizeLike with a zero matrix.

**0.15.363.4.32 conservativeResize() [2/2]** template<typename Settings>
static void smtrat::PBGaussModule<Settings>::conservativeResize(
 VectorT & v,
 long newSize) [inline], [static]

Convenience wrapper for Eigen::conservativeResizeLike with a zero vector.

**0.15.363.4.33 constraintsToInform()** const carl::FastSet<FormulaT>& smtrat::Module::constraints←
ToInform () const [inline], [inherited]

#### Returns

The constraints which the backends must be informed about.

**0.15.363.4.34 currentlySatisfied()** virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT & ) const [inline], [virtual], [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAbstractModule<Settings>](#), [smtrat::LRAbstractModule<LRASettingsICP>](#), and [smtrat::LRAbstractModule<LRASettings>](#).

**0.15.363.4.35 currentlySatisfiedByBackend()** unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & \_formula) const [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.363.4.36 deinform()** void smtrat::Module::deinform (
 const FormulaT & \_constraint) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.363.4.37 deinformCore()** virtual void smtrat::Module::deinformCore (
 const FormulaT & \_constraint) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only

sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings4 >](#).

**0.15.363.4.38 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const` [protected], [inherited]

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of `origins`

**0.15.363.4.39 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.363.4.40 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.363.4.41 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

Returns

**0.15.363.4.42 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.363.4.43 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

Returns

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.363.4.44 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.363.4.45 gaussAlgorithm()** `template<typename Settings > FormulaT smtrat::PBGaussModule< Settings >::gaussAlgorithm ()`

**0.15.363.4.46 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.363.4.47 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.363.4.48 getBackendsModel()** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.363.4.49 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.363.4.50 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.363.4.51 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.363.4.52 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.363.4.53 `getOrigins()` [1/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.363.4.54 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.363.4.55 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.363.4.56 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT >` `smtrat::Module::getReceivedFormulaSimplified( )` [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.363.4.57 `hasLemmas()`** `bool smtrat::Module::hasLemmas( )` [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.363.4.58 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const` [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.363.4.59 `id()`** `std::size_t smtrat::Module::id( ) const` [inline], [inherited]

**Returns**

A unique ID to identify this module instance.

**0.15.363.4.60 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const` [inline], [inherited]

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.363.4.61 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint )` [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.363.4.62 informBackends()** `void smrat::Module::informBackends (`

```
 const FormulaT & _constraint) [inline], [protected], [inherited]
```

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.363.4.63 informCore()** `template<typename Settings >`

```
bool smrat::PBGaussModule< Settings >::informCore (
```

```
 const FormulaT & _constraint) [virtual]
```

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.363.4.64 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informed→`

```
Constraints () const [inline], [inherited]
```

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.363.4.65 init()** `template<typename Settings >`

```
void smrat::PBGaussModule< Settings >::init () [virtual]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not and will not be called more than once and only before the first runBackends call.

Reimplemented from [smrat::Module](#).

**0.15.363.4.66 is\_minimizing()** `bool smrat::Module::is_minimizing ( ) const [inline], [inherited]`**0.15.363.4.67 isLemmaLevel()** `bool smrat::Module::isLemmaLevel (`

```
 LemmaLevel level) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

## Parameters

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.363.4.68 isPreprocessor()** `bool smtrat::Module::isPreprocessor () const [inline], [inherited]`

## Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.363.4.69 lemmas()** `const std::vector<Lemma>& smtrat::Module::lemmas () const [inline], [inherited]`

## Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.363.4.70 lookForReductionRow()** `template<typename Settings >  
std::vector<Rational> smtrat::PBGaussModule< Settings >::lookForReductionRow (  
 const MatrixT & uMatrix,  
 const VectorT & ineqRow,  
 long column )`**0.15.363.4.71 merge()** `std::vector< FormulaT > smtrat::Module::merge (`  
 `const std::vector< FormulaT > & _vecSetA,`  
 `const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

## Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

## Returns

The vector being the two given vectors merged.

**0.15.363.4.72 model()** `const Model& smtrat::Module::model () const [inline], [inherited]`

## Returns

The assignment of the current satisfiable result, if existent.

**0.15.363.4.73 modelsDisjoint()** `bool smtrat::Module::modelsDisjoint (`  
 `const Model & _modelA,`  
 `const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.363.4.74 `moduleName()`** `template<typename Settings >`  
`std::string smtrat::PBGaussModule< Settings >::moduleName ( ) const [inline], [virtual]`

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.363.4.75 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.363.4.76 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`  
`const FormulaT & _origin ) const [protected], [inherited]`

**0.15.363.4.77 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin (`  
`) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.363.4.78 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
`[inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.363.4.79 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.363.4.80 `pReceivedFormula()`** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.363.4.81 `print()`** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.363.4.82 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.363.4.83 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.363.4.84 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.363.4.85 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.363.4.86 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

---

**0.15.363.4.87 `probablyLooping()`** `bool smtrat::Module::probablyLooping (`  
 `const typename Poly::PolyType & _branchingPolynomial,`  
 `const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

`true`, if this branching is probably part of an infinite loop of branchings; `false`, otherwise.

**0.15.363.4.88 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.363.4.89 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset (`  
 `ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.363.4.90 `receivedVariable()`** `bool smtrat::Module::receivedVariable (`  
 `carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.363.4.91 `reconstructEqSystem()`** `template<typename Settings >`  
`FormulaT smtrat::PBGaussModule< Settings >::reconstructEqSystem (`  
 `const MatrixT & m,`  
 `const VectorT & b,`  
 `const carl::Variables & vars,`  
 `const std::vector< carl::Relation > & rels )`

**0.15.363.4.92 `reduce()`** `template<typename Settings >`  
`FormulaT smtrat::PBGaussModule< Settings >::reduce (`  
 `const MatrixT & u,`  
 `const VectorT & b,`  
 `const carl::Variables vars )`

**0.15.363.4.93 `remove()`** `void smtrat::Module::remove (`  
 `ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

**0.15.363.4.94 `removeCore()`** template<typename Settings >

```
void smrat::PBGaussModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

**Parameters**

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.363.4.95 `removeOrigin()`** std::pair<ModuleInput::iterator, bool> smrat::Module::remove←

Origin (

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.363.4.96 `removeOrigins()`** std::pair<ModuleInput::iterator, bool> smrat::Module::remove←

Origins (

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.363.4.97 `rPassedFormula()`** const ModuleInput& smrat::Module::rPassedFormula ( ) const

[inline], [inherited]

**Returns**

A reference to the formula passed to the backends of this module.

**0.15.363.4.98 `rReceivedFormula()`** const ModuleInput& smrat::Module::rReceivedFormula ( ) const

[inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.363.4.99 `runBackends()` [1/2]** virtual Answer smrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]

Reimplemented in [smrat::PModule](#).

---

**0.15.363.4.100 runBackends()** [2/2] [Answer](#) smtrat::Module::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.363.4.101 setId()** void smtrat::Module::setId (

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

**Parameters**

|                 |                                    |
|-----------------|------------------------------------|
| <code>id</code> | The id to set this module's id to. |
|-----------------|------------------------------------|

**0.15.363.4.102 setThreadPriority()** void smtrat::Module::setThreadPriority (

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.363.4.103 solverState()** [Answer](#) smtrat::Module::solverState ( ) const [inline], [inherited]

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.363.4.104 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint (

```
 const FormulaT & _unequalConstraint) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                             |
|---------------------------------|---------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol !=. |
|---------------------------------|---------------------------------------------|

**0.15.363.4.105 `threadPriority()`** `thread_priority` `smtrat::Module::threadPriority () const [inline], [inherited]`

**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.363.4.106 `updateAllModels()`** `void smtrat::Module::updateAllModels () [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in `smtrat::SATModule< Settings >`.

**0.15.363.4.107 `updateLemmas()`** `void smtrat::Module::updateLemmas () [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.363.4.108 `updateModel()`** `template<typename Settings > void smtrat::PBGaussModule< Settings >::updateModel () const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from `smtrat::Module`.

**0.15.363.4.109 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends () const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.363.5 Field Documentation

**0.15.363.5.1 `mAllBackends`** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.363.5.2 `mAllModels`** `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

**0.15.363.5.3 `mBackendsFoundAnswer`** `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.363.5.4 mConstraintsToInform** `carl::FastSet<FormulaT> smrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.363.5.5 mFinalCheck** `bool smrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.363.5.6 mFirstPosInLastBranches** `std::size_t smrat::Module::mFirstPosInLastBranches = 0` [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.363.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.363.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.363.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.363.5.10 mFullCheck** `bool smrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.363.5.11 mInfeasibleSubsets** `std::vector<FormulaSet> smrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.363.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.363.5.13 mLastBranches** `std::vector<Branching> smrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.363.5.14 mLemmas** `std::vector<Lemma> smrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.363.5.15 mModel** `Model smrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.363.5.16 mModelComputed** `bool smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.363.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.363.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.363.5.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.363.5.20 mpManager** `Manager* const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.363.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter` [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.363.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState` [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.363.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations` [protected], [inherited]

**0.15.363.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends` [protected], [inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.363.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters` [protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.364 smtrat::PBPPModule< Settings > Class Template Reference

```
#include <PBPPModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0, PERMANENT = 1 }`

## Public Member Functions

- std::string `moduleName () const`
- PBPPModule (const ModuleInput \*\_formula, Conditionals &\_conditionals, Manager \*\_manager=nullptr)
- ~PBPPModule ()
- bool `informCore (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- void `init ()`  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- bool `addCore (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- void `removeCore (ModuleInput::const_iterator _subformula)`  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- void `updateModel () const`  
*Updates the current assignment into the model.*
- Answer `checkCore ()`  
*Checks the received formula for consistency.*
- bool `inform (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- void `deinform (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- bool `add (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- virtual Answer `check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
*Checks the received formula for consistency.*
- virtual void `remove (ModuleInput::const_iterator _subformula)`  
*Removes everything related to the given sub-formula of the received formula.*
- virtual void `updateAllModels ()`  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > `getModelEqualities () const`  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned `currentlySatisfiedByBackend (const FormulaT &_formula) const`
- virtual unsigned `currentlySatisfied (const FormulaT &) const`
- bool `receivedVariable (carl::Variable::Arg _var) const`
- Answer `solverState () const`
- std::size\_t `id () const`
- void `setId (std::size_t _id)`  
*Sets this modules unique ID to identify itself.*
- thread\_priority `threadPriority () const`
- void `setThreadPriority (thread_priority _threadPriority)`  
*Sets the priority of this module to get a thread for running its check procedure.*
- const ModuleInput \* `pReceivedFormula () const`
- const ModuleInput & `rReceivedFormula () const`
- const ModuleInput \* `pPassedFormula () const`
- const ModuleInput & `rPassedFormula () const`
- const Model & `model () const`
- const std::vector< Model > & `allModels () const`
- const std::vector< FormulaSetT > & `infeasibleSubsets () const`
- const std::vector< Module \* > & `usedBackends () const`
- const carl::FastSet< FormulaT > & `constraintsToInform () const`
- const carl::FastSet< FormulaT > & `informedConstraints () const`

- void `addLemma` (const `FormulaT` &\_lemma, const `LemmaType` &\_lt=`LemmaType::NORMAL`, const `FormulaT` &\_preferredFormula=`FormulaT`(`carl::FormulaType::TRUE`))
 

*Stores a lemma being a valid formula.*
- bool `hasLemmas` ()
 

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas` ()
 

*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas` () const
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula` () const
- `ModuleInput::const_iterator firstSubformulaToPass` () const
- void `receivedFormulaChecked` ()
 

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals` & `answerFound` () const
- bool `isPreprocessor` () const
- `carl::Variable objective` () const
- bool `is_minimizing` () const
- void `excludeNotReceivedVariablesFromModel` () const
 

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas` ()
 

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations` ()
- void `collectOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
 

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- bool `isValidInfeasibleSubset` () const
- void `checkInfSubsetForMinimality` (std::vector< `FormulaSetT` >::const\_iterator \_insubset, const std::string &\_filename="smaller\_muses", unsigned \_maxSizeDifference=1) const
 

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual std::pair< bool, `FormulaT` > `getReceivedFormulaSimplified` ()
- void `print` (const std::string &\_initiation="\*\*\*") const
 

*Prints everything relevant of the solver.*
- void `printReceivedFormula` (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of the received formula.*
- void `printPassedFormula` (const std::string &\_initiation="\*\*\*") const
 

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const std::string &\_initiation="\*\*\*") const
 

*Prints the infeasible subsets.*
- void `printModel` (std::ostream &\_out=std::cout) const
 

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream &\_out=std::cout)
 

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual void `deinformCore` (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- ModuleInput::iterator `passedFormulaBegin` ()
- ModuleInput::iterator `passedFormulaEnd` ()
- void `addOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const FormulaT & `getOrigins` (ModuleInput::const\_iterator \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const FormulaT &\_formula, FormulasT &\_origins) const
- void `getOrigins` (const FormulaT &\_formula, FormulaSetT &\_origins) const
- std::pair< ModuleInput::iterator, bool > `removeOrigin` (ModuleInput::iterator \_formula, const FormulaT &\_origin)  
*ptr< std::vector< FormulaT >> &\_origins)*
- std::pair< ModuleInput::iterator, bool > `removeOrigins` (ModuleInput::iterator \_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)
- void `informBackends` (const FormulaT &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const FormulaT &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< ModuleInput::iterator, bool > `addReceivedSubformulaToPassedFormula` (ModuleInput::const\_iterator \_subformula)  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const FormulaT &\_origin) const
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula)  
*Adds the given formula to the passed formula with no origin.*
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)  
*Adds the given formula to the passed formula.*
- std::pair< ModuleInput::iterator, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula, const FormulaT &\_origin)  
*Adds the given formula to the passed formula.*

- void `generateTrivialInfeasibleSubset ()`  
*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`  
*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin (const std::vector< FormulaT > &_origins) const`
- void `getInfeasibleSubsets ()`  
*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets (const Module &backend) const`  
*Get the infeasible subsets the given backend provides.*
- const `Model & backendsModel () const`  
*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel () const`
- void `getBackendsAllModels () const`  
*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`  
*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`
- std::vector< `FormulaT` > `merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`  
*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin (const std::vector< FormulaT > &origins) const`
- bool `probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const`  
*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`  
*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &&_premise=std::vector< FormulaT >())`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())`
- void `splitUnequalConstraint (const FormulaT &)`  
*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel () const`
- void `addInformationRelevantFormula (const FormulaT &formula)`  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas ()`  
*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel (LemmaLevel level)`  
*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector<FormulaSetT> mInfeasibleSubsets`

*Stores the infeasible subsets.*

- `Manager *const mpManager`

*A reference to the manager.*

- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*

- `std::vector<Model> mAllModels`

*Stores all satisfying assignments.*

- `bool mModelComputed`

*True, if the model has already been computed.*

- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*

- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*

- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*

- `std::vector<TheoryPropagation> mTheoryPropagations`

- `std::atomic<Answer> mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- `Conditionals mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- `std::vector<Module*> mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- `std::vector<Module*> mAllBackends`

*The backends of this module which have been used.*

- `std::vector<Lemma> mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*

- `ModuleInput::iterator mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- `carl::FastSet<FormulaT> mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet<FormulaT> mInformedConstraints`

*Stores the position of the first constraint of which no backend has been informed about.*

- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- `unsigned mSmallerMusesCheckCounter`

*Counter used for the generation of the smt2 files to check for smaller muses.*

- `std::vector<std::size_t> mVariableCounters`

*Maps variables to the number of their occurrences.*

### 0.15.364.1 Member Typedef Documentation

**0.15.364.1.1 SettingsType** template<typename Settings >  
 typedef `Settings smrat::PBPPModule< Settings >::SettingsType`

### 0.15.364.2 Member Enumeration Documentation

**0.15.364.2.1 LemmaType** enum `smrat::Module::LemmaType` : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.364.3 Constructor & Destructor Documentation

**0.15.364.3.1 PBPPModule()** template<typename Settings >  
`smrat::PBPPModule< Settings >::PBPPModule` (  
   const `ModuleInput` \* `_formula`,  
   `Conditionals` & `_conditionals`,  
   `Manager` \* `_manager` = `nullptr` )

**0.15.364.3.2 ~PBPPModule()** template<typename Settings >  
`smrat::PBPPModule< Settings >::~PBPPModule` ( )

### 0.15.364.4 Member Function Documentation

**0.15.364.4.1 add()** bool `smrat::Module::add` (  
   `ModuleInput::const_iterator` `_subformula` ) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.364.4.2 addConstraintToInform()** void `smrat::Module::addConstraintToInform` (  
   const `FormulaT` & `_constraint` ) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                    |                        |
|--------------------|------------------------|
| <i>_constraint</i> | The constraint to add. |
|--------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.364.4.3 addCore()** `template<typename Settings >`

```
bool smtrat::PBPPModule< Settings >::addCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub-formula to take additionally into account. |
|--------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.364.4.4 addInformationRelevantFormula()** `void smtrat::Module::addInformationRelevantFormula (`

```
 const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.364.4.5 addLemma()** `void smtrat::Module::addLemma (`

```
 const FormulaT & _lemma,
```

```
 const LemmaType & _lt = LemmaType::NORMAL,
```

```
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
```

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.364.4.6 addOrigin()** `void smtrat::Module::addOrigin (`

```
 ModuleInput::iterator _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.364.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addReceivedSubformulaToPassedFormula (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.364.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.364.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.364.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.364.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline],
[inherited]
```

#### Returns

All satisfying assignments, if existent.

```
0.15.364.4.12 anAnswerFound() bool smrat::Module::anAnswerFound () const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

```
0.15.364.4.13 answerFound() const smrat::Conditionals& smrat::Module::answerFound () const
[inline], [inherited]
```

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

```
0.15.364.4.14 backendsModel() const Model & smrat::Module::backendsModel () const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

```
0.15.364.4.15 branchAt() [1/4] bool smtrat::Module::branchAt (
 carl::Variable::Arg _var,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

```
0.15.364.4.16 branchAt() [2/4] bool smtrat::Module::branchAt (
 carl::Variable::Arg _var,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

```
0.15.364.4.17 branchAt() [3/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

```
0.15.364.4.18 branchAt() [4/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

---

**0.15.364.4.19 check()** *Answer* smrat::Module::check (

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smrat::PModule](#).

**0.15.364.4.20 checkCore()** *template<typename Settings >*  
*Answer* smrat::PBPModule< Settings >::checkCore ( ) [virtual]

Checks the received formula for consistency.

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.364.4.21 checkInfSubsetForMinimality()** *void* smrat::Module::checkInfSubsetForMinimality (

```
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.364.4.22 checkModel()** *unsigned* smrat::Module::checkModel ( ) const [protected], [inherited]

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.364.4.23 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.364.4.24 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.364.4.25 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.364.4.26 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.364.4.27 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.364.4.28 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.364.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.364.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.364.4.31 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.364.4.32 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.364.4.33 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.364.4.34 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.364.4.35 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.364.4.36 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.364.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

```
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.364.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.364.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`

```
 const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns****0.15.364.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.364.4.41 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.364.4.42 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.364.4.43 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.364.4.44 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.364.4.45 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.364.4.46 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.364.4.47 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.364.4.48 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.364.4.49 getModelEqualities()** std::list< std::vector< `carl::Variable` > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.364.4.50 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.364.4.51 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.364.4.52 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]  
Gets the origins of the passed formula at the given position.

#### Parameters

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.364.4.53 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.364.4.54 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]  
Checks whether this module or any of its backends provides any lemmas.

**0.15.364.4.55 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

#### Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.364.4.56 `id()`** `std::size_t smtrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.364.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.364.4.58 `inform()`** `bool smtrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.364.4.59 `informBackends()`** `void smtrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.364.4.60 `informCore()`** `template<typename Settings > bool smtrat::PBPPModule< Settings >::informCore (const FormulaT & _constraint) [virtual]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.364.4.61 informedConstraints()** const carl::FastSet<[FormulaT](#)>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.364.4.62 init()** template<typename Settings >  
void smtrat::PBPModule< Settings >::init ( ) [virtual]

Informs all backends about the so far encountered constraints, which have not yet been communicated.  
This method must not and will not be called more than once and only before the first runBackends call.  
Reimplemented from [smtrat::Module](#).

**0.15.364.4.63 is\_minimizing()** bool smtrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.364.4.64 isLemmaLevel()** bool smtrat::Module::isLemmaLevel ( [LemmaLevel](#) level ) [protected], [inherited]

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.364.4.65 isPreprocessor()** bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.364.4.66 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.364.4.67 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & \_vecSetA, const std::vector< [FormulaT](#) > & \_vecSetB ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.364.4.68 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.364.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.364.4.70 `moduleName()`** `template<typename Settings > std::string smtrat::PBPPModule< Settings >::moduleName () const [inline], [virtual]`**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.364.4.71 `objective()`** `carl::Variable smtrat::Module::objective () const [inline], [inherited]`**0.15.364.4.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (const FormulaT & _origin ) const [protected], [inherited]`**0.15.364.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin () [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.364.4.74 passedFormulaEnd()** `ModuleInput::iterator smrat::Module::passedFormulaEnd () [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.364.4.75 pPassedFormula()** `const ModuleInput* smrat::Module::pPassedFormula () const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.364.4.76 pReceivedFormula()** `const ModuleInput* smrat::Module::pReceivedFormula () const [inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.364.4.77 print()** `void smrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.364.4.78 printAllModels()** `void smrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.364.4.79 printInfeasibleSubsets()** `void smrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const [inherited]`

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.364.4.80 printModel()** `void smrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.364.4.81 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.364.4.82 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.364.4.83 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.364.4.84 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.364.4.85 `receivedFormulasAsInfeasibleSubset()`** `void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Stores an infeasible subset consisting only of the given received formula.

**0.15.364.4.86 receivedVariable()** `bool smrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.364.4.87 remove()** `void smrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smrat::PModule](#).

**0.15.364.4.88 removeCore()** `template<typename Settings > void smrat::PBPPModule< Settings >::removeCore ( ModuleInput::const_iterator _subformula ) [virtual]`

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.364.4.89 removeOrigin()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.364.4.90 removeOrigins()** `std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

**0.15.364.4.91 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const [inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.364.4.92 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A reference to the formula passed to this module.

**0.15.364.4.93 runBackends() [1/2]** `virtual Answer smtrat::Module::runBackends () [inline], [protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.364.4.94 runBackends() [2/2]** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.364.4.95 setId()** `void smtrat::Module::setId (`

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

**Parameters**

|                   |                                    |
|-------------------|------------------------------------|
| <code>↔ id</code> | The id to set this module's id to. |
|-------------------|------------------------------------|

**0.15.364.4.96 setThreadPriority()** `void smtrat::Module::setThreadPriority (`

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.364.4.97 solverState()** `Answer smtrat::Module::solverState () const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.364.4.98 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.364.4.99 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.364.4.100 updateAllModels()** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.364.4.101 updateLemmas()** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.364.4.102 updateModel()** `template<typename Settings >`

`void smtrat::PBPPModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.364.4.103 usedBackends()** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.364.5 Field Documentation****0.15.364.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.364.5.2 mAllModels** `std::vector<Model>` `smtrat::Module::mAllModels` [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.364.5.3 mBackendsFoundAnswer** `std::atomic_bool*` `smtrat::Module::mBackendsFoundAnswer` [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.364.5.4 mConstraintsToInform** `carl::FastSet<FormulaT>` `smtrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.364.5.5 mFinalCheck** `bool` `smtrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.364.5.6 mFirstPosInLastBranches** `std::size_t` `smtrat::Module::mFirstPosInLastBranches` = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.364.5.7 mFirstSubformulaToPass** `ModuleInput::iterator` `smtrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.364.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator` `smtrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.364.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.364.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.364.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.364.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.364.5.13 mLastBranches** std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.364.5.14 mLemmas** std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]

Stores the lemmas being valid formulas this module or its backends made.

**0.15.364.5.15 mModel** Model smtrat::Module::mModel [mutable], [protected], [inherited]

Stores the assignment of the current satisfiable result, if existent.

**0.15.364.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]

True, if the model has already been computed.

**0.15.364.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore =

5 [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.364.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]

Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.364.5.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]

Reusable splitting variables.

**0.15.364.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]

A reference to the manager.

**0.15.364.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]

Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.364.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.364.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.364.5.24 mUsedBackends** std::vector<[Module](#)\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.364.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.365 smtrat::PersistentUnionFind Struct Reference

```
#include <UnionFind.h>
```

### Public Types

- using [Value](#) = size\_t
- using [Representative](#) = [Value](#)
- using [Parents](#) = [immer](#)::vector< [Value](#) >
- using [Ranks](#) = [immer](#)::vector< size\_t >
- using [FindState](#) = std::pair< [Parents](#), [Representative](#) >

### Public Member Functions

- void [update\\_parents](#) ([Parents](#) const &parents) noexcept
- void [update\\_parents](#) ([Parents](#) &&parents) noexcept
- void [update\\_ranks](#) ([Ranks](#) const &ranks) noexcept
- void [update\\_ranks](#) ([Ranks](#) &&ranks) noexcept
- template<typename Ps, typename Rs >  
void [update](#) (Ps &&parents, Rs &&ranks) noexcept
- void [resize](#) (size\_t size) noexcept
- void [init\\_size](#) (size\_t size) noexcept
- [FindState](#) [find\\_impl](#) ([Parents](#) parents, [Value](#) const &i) const noexcept
- [Representative](#) [find](#) ([Value](#) const &val) noexcept
- [PersistentUnionFind](#) [merge](#) ([Value](#) const &a, [Value](#) const &b) noexcept
- void [dump](#) () const noexcept

### Data Fields

- [Parents \\_parents](#)
- [Ranks \\_ranks](#)

### 0.15.365.1 Member Typedef Documentation

**0.15.365.1.1 FindState** using smtrat::PersistentUnionFind::FindState = std::pair<[Parents](#), [Representative](#)>**0.15.365.1.2 Parents** using smtrat::PersistentUnionFind::Parents = [immer](#)::vector<[Value](#)>**0.15.365.1.3 Ranks** using smtrat::PersistentUnionFind::Ranks = [immer](#)::vector<size\_t>**0.15.365.1.4 Representative** using smtrat::PersistentUnionFind::Representative = [Value](#)

**0.15.365.1.5 Value** using `smtrat::PersistentUnionFind::Value = size_t`

### 0.15.365.2 Member Function Documentation

**0.15.365.2.1 dump()** `void smtrat::PersistentUnionFind::dump() const [inline], [noexcept]`

**0.15.365.2.2 find()** `Representative smtrat::PersistentUnionFind::find( Value const & val ) [inline], [noexcept]`

**0.15.365.2.3 find\_impl()** `FindState smtrat::PersistentUnionFind::find_impl( Parents parents, Value const & i ) const [inline], [noexcept]`

**0.15.365.2.4 init\_size()** `void smtrat::PersistentUnionFind::init_size( size_t size ) [inline], [noexcept]`

**0.15.365.2.5 merge()** `PersistentUnionFind smtrat::PersistentUnionFind::merge( Value const & a, Value const & b ) [inline], [noexcept]`

**0.15.365.2.6 resize()** `void smtrat::PersistentUnionFind::resize( size_t size ) [inline], [noexcept]`

**0.15.365.2.7 update()** `template<typename Ps, typename Rs> void smtrat::PersistentUnionFind::update( Ps && parents, Rs && ranks ) [inline], [noexcept]`

**0.15.365.2.8 update\_parents() [1/2]** `void smtrat::PersistentUnionFind::update_parents( Parents && parents ) [inline], [noexcept]`

**0.15.365.2.9 update\_parents() [2/2]** `void smtrat::PersistentUnionFind::update_parents( Parents const & parents ) [inline], [noexcept]`

**0.15.365.2.10 update\_ranks() [1/2]** `void smtrat::PersistentUnionFind::update_ranks( Ranks && ranks ) [inline], [noexcept]`

**0.15.365.2.11 update\_ranks() [2/2]** `void smtrat::PersistentUnionFind::update_ranks( Ranks const & ranks ) [inline], [noexcept]`

### 0.15.365.3 Field Documentation

---

**0.15.365.3.1 `_parents`** `Parents smtrat::PersistentUnionFind::_parents`

**0.15.365.3.2 `_ranks`** `Ranks smtrat::PersistentUnionFind::_ranks`

## 0.15.366 `smtrat::PFEModule`< `Settings` > Class Template Reference

```
#include <PFEModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `PFEModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=NULL)`
- `~PFEModule ()`
- `bool addCore (ModuleInput::const_iterator)`

*The module has to take the given sub-formula of the received formula into account.*
- `Answer checkCore ()`

*Checks the received formula for consistency.*
- `void removeCore (ModuleInput::const_iterator)`

*Removes everything related to the given sub-formula of the received formula.*
- `bool isPreprocessor () const`
- `bool appliedPreprocessing () const`
- `bool add (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- `void remove (ModuleInput::const_iterator _subformula)`

*Removes everything related to the given sub-formula of the received formula.*
- `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`

*Checks the received formula for consistency.*
- `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*
- `virtual Answer runBackends ()`
- `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void updateModel () const`

*Updates the current assignment into the model.*
- `bool inform (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- `void deinform (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- `virtual void init ()`

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `virtual void updateAllModels ()`

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
- `virtual unsigned currentlySatisfied (const FormulaT &) const`
- `bool receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- `std::size_t id () const`

- void `setId` (std::size\_t \_id)
 

*Sets this module's unique ID to identify itself.*
- `thread_priority threadPriority () const`
- void `setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput * pReceivedFormula () const`
- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const std::vector< `Model` > & `allModels () const`
- const std::vector< `FormulaSetT` > & `infeasibleSubsets () const`
- const std::vector< `Module` \* > & `usedBackends () const`
- const carl::FastSet< `FormulaT` > & `constraintsToInform () const`
- const carl::FastSet< `FormulaT` > & `informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- virtual std::string `moduleName () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- bool `isValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`

- Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.
- void `printAllModels` (std::ostream &\_out=std::cout)  
Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

## Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` &\_splittingVariable)

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector<`Branching`> `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector<`FormulaT`> `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair<`ModuleInput::iterator`, bool> `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)  
*Removes the origin of the given formula from the passed formula.*
- std::pair<`ModuleInput::iterator`, bool> `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr<std::vector<`FormulaT`>> &\_origins)  
*Removes all origins of the given formula from the passed formula.*
- void `informBackends` (const `FormulaT` &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*

- std::pair< [ModuleInput::iterator](#), bool > [addReceivedSubformulaToPassedFormula](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool [originInReceivedFormula](#) (const [FormulaT](#) &\_origin) const
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const std::shared\_ptr< std::vector< [FormulaT](#) >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
- void [getInfeasibleSubsets](#) ()
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void [getBackendsModel](#) () const
- void [getBackendsAllModels](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
- virtual [ModuleInput::iterator](#) [eraseSubformulaFromPassedFormula](#) ([ModuleInput::iterator](#) \_subformula, bool \_ignoreOrigins=false)
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void [clearPassedFormula](#) ()
 

*Clears the passed formula.*
- std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const
 

*Merges the two vectors of sets into the first one.*
- size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &\_origins) const
- bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool [branchAt](#) (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- void [splitUnequalConstraint](#) (const [FormulaT](#) &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned [checkModel](#) () const

- void `addInformationRelevantFormula` (const `FormulaT` &formula)  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()  
*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (`LemmaLevel` level)  
*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)  
*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- `Manager` \*const `mpManager`  
*A reference to the manager.*
- `Model` `mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`  
*Stores all satisfying assignments.*
- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals` `mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`  
*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator` `mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- carl::FastSet< `FormulaT` > `mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator` `mFirstUncheckedReceivedSubformula`

- **unsigned mSmallerMusesCheckCounter**  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- **std::vector< std::size\_t > mVariableCounters**  
*Maps variables to the number of their occurrences.*

### 0.15.366.1 Member Typedef Documentation

**0.15.366.1.1 SettingsType** template<typename Settings >  
**typedef** Settings smrat::PFEModule< Settings >::SettingsType

### 0.15.366.2 Member Enumeration Documentation

**0.15.366.2.1 LemmaType** enum smrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.366.3 Constructor & Destructor Documentation

**0.15.366.3.1 PFEModule()** template<typename Settings >  
smrat::PFEModule< Settings >::PFEModule ( const ModuleInput \* \_formula, Conditionals & \_conditionals, Manager \* \_manager = NULL )

**0.15.366.3.2 ~PFEModule()** template<typename Settings >  
smrat::PFEModule< Settings >::~PFEModule ( )

### 0.15.366.4 Member Function Documentation

**0.15.366.4.1 add()** bool smrat::PModule::add ( ModuleInput::const\_iterator \_subformula ) [inherited]  
The module has to take the given sub-formula of the received formula into account.

Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.366.4.2 addConstraintToInform()** void smtrat::Module::addConstraintToInform ( const FormulaT & \_constraint ) [protected], [virtual], [inherited]  
Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

Parameters

|             |                        |
|-------------|------------------------|
| _constraint | The constraint to add. |
|-------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.366.4.3 addCore()** template<typename Settings >

bool smtrat::PFEModule< Settings >::addCore ( ModuleInput::const\_iterator formula ) [virtual]

The module has to take the given sub-formula of the received formula into account.

Parameters

|         |                                                    |
|---------|----------------------------------------------------|
| formula | The sub-formula to take additionally into account. |
|---------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.366.4.4 addInformationRelevantFormula()** void smtrat::Module::addInformationRelevantFormula

(

const FormulaT & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

Parameters

|         |                |
|---------|----------------|
| formula | Formula to add |
|---------|----------------|

**0.15.366.4.5 addLemma()** void smtrat::Module::addLemma (

const FormulaT & \_lemma,

const LemmaType & \_lt = LemmaType::NORMAL,

const FormulaT & \_preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],

[inherited]

Stores a lemma being a valid formula.

Parameters

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| _lemma            | The eduction/lemma to store.                                          |
| _lt               | The type of the lemma.                                                |
| _preferredFormula | Hint for the next decision, which formula should be assigned to true. |

**0.15.366.4.6 addOrigin()** void smtrat::Module::addOrigin (

ModuleInput::iterator \_formula,

```
const FormulaT & _origin) [inline], [protected], [inherited]
Adds the given set of formulas in the received formula to the origins of the given passed formula.
```

#### Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.366.4.7 addReceivedSubformulaToPassedFormula()** std::pair<[ModuleInput::iterator](#),bool>  
`smtrat::Module::addReceivedSubformulaToPassedFormula (`

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.366.4.8 addSubformulaToPassedFormula() [1/3]** std::pair<[ModuleInput::iterator](#),bool> smtrat<->  
`::Module::addSubformulaToPassedFormula (`

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.366.4.9 addSubformulaToPassedFormula() [2/3]** std::pair<[ModuleInput::iterator](#),bool> smtrat<->  
`::Module::addSubformulaToPassedFormula (`

```
 const FormulaT & _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.366.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>**

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.366.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]****Returns**

All satisfying assignments, if existent.

**0.15.366.4.12 anAnswerFound() bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.366.4.13 answerFound() const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.366.4.14 appliedPreprocessing() bool smrat::PModule::appliedPreprocessing () const [inline], [inherited]****Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.366.4.15 `backendsModel()`** const `Model` & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.366.4.16 `branchAt() [1/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.366.4.17 `branchAt() [2/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.366.4.18 `branchAt() [3/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.366.4.19 `branchAt() [4/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [protected], [inherited]

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                            |                                                                                                                                                                                                                |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>   | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>     | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>        | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>      | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code> | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |

**Parameters**

|                                           |                                                                                              |
|-------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise. |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                |

**0.15.366.4.20 `check()`** `Answer smtrat::PModule::check (`

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.366.4.21 `checkCore()`** `template<typename Settings >`

```
Answer smtrat::PFEModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

**Returns**

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.366.4.22 `checkInfeasibleSubsetForMinimality()`** `void smtrat::Module::checkInfeasibleSubsetForMinimality (`

```
 std::vector< FormulaSet >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.366.4.23 `checkModel()`** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.366.4.24 `cleanModel()`** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.366.4.25 `clearLemmas()`** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.366.4.26 `clearModel()`** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.366.4.27 `clearModels()`** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.366.4.28 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.366.4.29 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.366.4.30 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.366.4.31 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

**0.15.366.4.32 constraintsToInform()** const carl::FastSet<[FormulaT](#)>& smrat::Module::constraintsToInform( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.366.4.33 currentlySatisfied()** virtual unsigned smrat::Module::currentlySatisfied( const [FormulaT](#) & ) const [inline], [virtual], [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.366.4.34 currentlySatisfiedByBackend()** unsigned smrat::Module::currentlySatisfiedByBackend( const [FormulaT](#) & \_formula ) const [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.366.4.35 deinform()** void smrat::Module::deinform( const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.366.4.36 deinformCore()** virtual void smrat::Module::deinformCore( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.366.4.37 determine\_smallest\_origin()** size\_t smrat::Module::determine\_smallest\_origin( const std::vector< [FormulaT](#) > & origins ) const [protected], [inherited]

## Parameters

|                       |                              |
|-----------------------|------------------------------|
| <code>_origins</code> | A vector of sets of origins. |
|-----------------------|------------------------------|

## Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.366.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

```
ModuleInput::iterator _subformula,
bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

## Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

## Returns

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.366.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.366.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`

```
const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

## Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

## Returns

**0.15.366.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.366.4.42 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.366.4.43 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.366.4.44 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.366.4.45 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.366.4.46 getBackendsModel()** `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.366.4.47 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.366.4.48 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (`

`const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.366.4.49 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.366.4.50 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.366.4.51 `getOrigins() [1/3]`** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulaSetT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.366.4.52 `getOrigins() [2/3]`** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulasT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.366.4.53 `getOrigins() [3/3]`** `const FormulaT& smtrat::Module::getOrigins (`

```
ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.366.4.54 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.366.4.55 hasLemmas()** `bool smrat::Module::hasLemmas () [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.366.4.56 hasValidInfeasibleSubset()** `bool smrat::Module::hasValidInfeasibleSubset () const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.366.4.57 id()** `std::size_t smrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.366.4.58 infeasibleSubsets()** `const std::vector<FormulaSetT>& smrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.366.4.59 inform()** `bool smrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.366.4.60 informBackends()** `void smrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.366.4.61 informCore()** `virtual bool smrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

`false`, if it can be easily decided whether the given constraint is inconsistent; `true`, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

#### 0.15.366.4.62 informedConstraints()

```
const carl::FastSet<FormulaT>& smrat::Module::informed<→
Constraints () const [inline], [inherited]
```

#### Returns

The position of the first constraint of which no backend has been informed about.

#### 0.15.366.4.63 init()

```
void smrat::Module::init () [virtual], [inherited]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

#### 0.15.366.4.64 is\_minimizing()

```
bool smrat::Module::is_minimizing () const [inline], [inherited]
```

#### 0.15.366.4.65 isLemmaLevel()

```
bool smrat::Module::isLemmaLevel (
 LemmaLevel level) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

#### 0.15.366.4.66 isPreprocessor()

```
bool smrat::PModule::isPreprocessor () const [inline], [inherited]
```

#### Returns

`true`, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.366.4.67 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.366.4.68 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & [\\_vecSetA](#), const std::vector< [FormulaT](#) > & [\\_vecSetB](#) ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                          |                                  |
|--------------------------|----------------------------------|
| <a href="#">_vecSetA</a> | A vector of sets of constraints. |
| <a href="#">_vecSetB</a> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.366.4.69 model()** const [Model](#)& smtrat::Module::model ( ) const [inline], [inherited]

**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.366.4.70 modelsDisjoint()** bool smtrat::Module::modelsDisjoint ( const [Model](#) & [\\_modelA](#), const [Model](#) & [\\_modelB](#) ) [static], [protected], [inherited]

Checks whether there is no variable assigned by both models.

**Parameters**

|                         |                                |
|-------------------------|--------------------------------|
| <a href="#">_modelA</a> | The first model to check for.  |
| <a href="#">_modelB</a> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.366.4.71 moduleName()** virtual std::string smtrat::Module::moduleName ( ) const [inline], [virtual], [inherited]

**Returns**

The name of the given module type as name.

Reimplemented in `smrat::VSModule< Settings >`, `smrat::UnionFindModule< Settings >`, `smrat::UFCegarModule< Settings >`, `smrat::STropModule< Settings >`, `smrat::SplitSOSModule< Settings >`, `smrat::PBPPModule< Settings >`, `smrat::PBGaussModule< Settings >`, `smrat::NRAILModule< Settings >`, `smrat::NewCADModule< Settings >`, `smrat::LVEModule< Settings >`, `smrat::LRAModule< Settings >`, `smrat::LRAModule< LRASettingsICP >`, `smrat::LRAModule< LRASettings1 >`, `smrat::IntEqModule< Settings >`, `smrat::IntBlastModule< Settings >`, `smrat::IncWidthModule< Settings >`, `smrat::ICPModule< Settings >`, `smrat::ICPModule< ICPSettings4 >`, `smrat::ICPModule< ICPSettings1 >`, `smrat::GBModule< Settings >`, `smrat::FouMoModule< Settings >`, `smrat::EQPreprocessingModule< Settings >`, `smrat::EQModule< Settings >`, `smrat::CurryModule< Settings >`, `smrat::CubeLIAModule< Settings >`, `smrat::CSplitModule< Settings >`, `smrat::BVMModule< Settings >`, and `smrat::BEModule< Settings >`.

**0.15.366.4.72 `objective()`** `carl::Variable smrat::Module::objective ( ) const [inline], [inherited]`**0.15.366.4.73 `originInReceivedFormula()`** `bool smrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`**0.15.366.4.74 `passedFormulaBegin()`** `ModuleInput::iterator smrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.366.4.75 `passedFormulaEnd()`** `ModuleInput::iterator smrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula.

**0.15.366.4.76 `pPassedFormula()`** `const ModuleInput* smrat::Module::pPassedFormula ( ) const [inline], [inherited]`**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.366.4.77 `pReceivedFormula()`** `const ModuleInput* smrat::Module::pReceivedFormula ( ) const [inline], [inherited]`**Returns**

A pointer to the formula passed to this module.

**0.15.366.4.78 `print()`** `void smrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.366.4.79 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.366.4.80 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.366.4.81 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.366.4.82 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.366.4.83 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.366.4.84 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.366.4.85 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline], [inherited]
```

Notifies that the received formulas has been checked.

```
0.15.366.4.86 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs<= InfeasibleSubset (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.366.4.87 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.366.4.88 remove() void smtrat::PModule::remove (
```

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub formula of the received formula to remove. |
|--------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

```
0.15.366.4.89 removeCore() template<typename Settings >
void smtrat::PFEModule< Settings >::removeCore (
 ModuleInput::const_iterator formula) [virtual]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>formula</i> | The sub formula of the received formula to remove. |
|----------------|----------------------------------------------------|

Reimplemented from [smrat::Module](#).

**0.15.366.4.90 removeOrigin()** `std::pair<ModuleInput::iterator, bool> smrat::Module::removeOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.366.4.91 removeOrigins()** `std::pair<ModuleInput::iterator, bool> smrat::Module::removeOrigins (`

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.366.4.92 rPassedFormula()** `const ModuleInput& smrat::Module::rPassedFormula ( ) const`  
`[inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.366.4.93 rReceivedFormula()** `const ModuleInput& smrat::Module::rReceivedFormula ( ) const`  
`[inline], [inherited]`

#### Returns

A reference to the formula passed to this module.

**0.15.366.4.94 runBackends() [1/2]** `virtual Answer smrat::PModule::runBackends ( ) [inline],`  
`[virtual], [inherited]`

Reimplemented from [smrat::Module](#).

**0.15.366.4.95 runBackends() [2/2]** `Answer smrat::PModule::runBackends (`

|                                                                 |                                                                                                                                       |
|-----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>bool _final,</code>                                       | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>bool _full,</code>                                        | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>carl::Variable _objective ) [virtual], [inherited]</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.366.4.96 setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]  
 Sets this modules unique ID to identify itself.

#### Parameters

|                                                 |                                    |
|-------------------------------------------------|------------------------------------|
| $\leftarrow$<br>$\overline{\leftarrow}$<br>$id$ | The id to set this module's id to. |
|-------------------------------------------------|------------------------------------|

**0.15.366.4.97 setThreadPriority()** void smtrat::Module::setThreadPriority ( thread\_priority \_threadPriority ) [inline], [inherited]  
 Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| $_threadPriority$ | The priority to set this module's thread priority to. |
|-------------------|-------------------------------------------------------|

**0.15.366.4.98 solverState()** Answer smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.366.4.99 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]  
 Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                      |                                                  |
|----------------------|--------------------------------------------------|
| $_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|----------------------|--------------------------------------------------|

**0.15.366.4.100 threadPriority()** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.366.4.101 updateAllModels()** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]  
 Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
 Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.366.4.102 updateLemmas()** void smtrat::Module::updateLemmas ( ) [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.366.4.103 updateModel()** void smtrat::PModule::updateModel ( ) const [virtual], [inherited]  
Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.366.4.104 usedBackends()** const std::vector<[Module](#)\*>& smtrat::Module::usedBackends ( )  
const [inline], [inherited]

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.366.5 Field Documentation

**0.15.366.5.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

**0.15.366.5.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.366.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.366.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.366.5.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.366.5.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.366.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.366.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.366.5.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.366.5.10 mFullCheck** `bool smtrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.366.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`  
Stores the infeasible subsets.

**0.15.366.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.366.5.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`  
Stores the last branches in a cycle buffer.

**0.15.366.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.366.5.15 mModel** `Model smtrat::Module::mModel [mutable], [protected], [inherited]`  
Stores the assignment of the current satisfiable result, if existent.

**0.15.366.5.16 mModelComputed** `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`  
True, if the model has already been computed.

**0.15.366.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.366.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.366.5.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables [static], [inherited]`  
Reusable splitting variables.

**0.15.366.5.20 mpManager** `Manager* const smrat::Module::mpManager [protected], [inherited]`  
A reference to the manager.

**0.15.366.5.21 mSmallerMusesCheckCounter** `unsigned smrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.366.5.22 mSolverState** `std::atomic<Answer> smrat::Module::mSolverState [protected], [inherited]`  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.366.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smrat::Module::mTheoryPropagations [protected], [inherited]`

**0.15.366.5.24 mUsedBackends** `std::vector<Module*> smrat::Module::mUsedBackends [protected], [inherited]`  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.366.5.25 mVariableCounters** `std::vector<std::size_t> smrat::Module::mVariableCounters [protected], [inherited]`  
Maps variables to the number of their occurrences.

## 0.15.367 smrat::PModule Class Reference

```
#include <PModule.h>
```

### Public Types

- enum class `LemmaType` : unsigned { `NORMAL` = 0 , `PERMANENT` = 1 }

### Public Member Functions

- `PModule` (const `ModuleInput` \*`_formula`, `Conditionals` &`_foundAnswer`, `Manager` \*`_manager=nullptr`, std::string `module_name="PModule"`)
- `bool isPreprocessor () const`
- `bool appliedPreprocessing () const`
- `bool add (ModuleInput::const_iterator _subformula)`  
*The module has to take the given sub-formula of the received formula into account.*
- `void remove (ModuleInput::const_iterator _subformula)`  
*Removes everything related to the given sub-formula of the received formula.*
- `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
*Checks the received formula for consistency.*
- `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- `virtual Answer runBackends ()`
- `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void updateModel () const`  
*Updates the current assignment into the model.*
- `bool inform (const FormulaT &_constraint)`

- **Informs the module about the given constraint.**
- void **deinform** (const **FormulaT** &\_constraint)
  - The inverse of informing about a constraint.*
- virtual void **init** ()
  - Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- virtual void **updateAllModels** ()
  - Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const
  - Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned **currentlySatisfiedByBackend** (const **FormulaT** &\_formula) const
- virtual unsigned **currentlySatisfied** (const **FormulaT** &) const
- bool **receivedVariable** (carl::Variable::Arg \_var) const
- **Answer solverState** () const
- std::size\_t **id** () const
- void **setId** (std::size\_t \_id)
  - Sets this modules unique ID to identify itself.*
- **thread\_priority threadPriority** () const
- void **setThreadPriority** (**thread\_priority** \_threadPriority)
  - Sets the priority of this module to get a thread for running its check procedure.*
- const **ModuleInput** \* **pReceivedFormula** () const
- const **ModuleInput** & **rReceivedFormula** () const
- const **ModuleInput** \* **pPassedFormula** () const
- const **ModuleInput** & **rPassedFormula** () const
- const **Model** & **model** () const
- const std::vector< **Model** > & **allModels** () const
- const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
- const std::vector< **Module** \* > & **usedBackends** () const
- const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
- const carl::FastSet< **FormulaT** > & **informedConstraints** () const
- void **addLemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lt= **LemmaType**::NORMAL, const **FormulaT** &\_preferredFormula= **FormulaT**(carl::FormulaType::TRUE))
  - Stores a lemma being a valid formula.*
- bool **hasLemmas** ()
  - Checks whether this module or any of its backends provides any lemmas.*
- void **clearLemmas** ()
  - Deletes all yet found lemmas.*
- const std::vector< **Lemma** > & **lemmas** () const
- **ModuleInput**::const\_iterator **firstUncheckedReceivedSubformula** () const
- **ModuleInput**::const\_iterator **firstSubformulaToPass** () const
- void **receivedFormulaChecked** ()
  - Notifies that the received formulas has been checked.*
- const **smrat::Conditionals** & **answerFound** () const
- virtual std::string **moduleName** () const
- carl::Variable **objective** () const
- bool **is\_minimizing** () const
- void **excludeNotReceivedVariablesFromModel** () const
  - Excludes all variables from the current model, which do not occur in the received formula.*
- void **updateLemmas** ()
  - Stores all lemmas of any backend of this module in its own lemma vector.*
- void **collectTheoryPropagations** ()
- void **collectOrigins** (const **FormulaT** &\_formula, **FormulasT** &\_origins) const
  - Collects the formulas in the given formula, which are part of the received formula.*
- void **collectOrigins** (const **FormulaT** &\_formula, **FormulaSetT** &\_origins) const

- bool `isValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _insubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore = 5`  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches = 0`  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual bool `informCore (const FormulaT &_constraint)`  
*Informs the module about the given constraint.*
- virtual void `deinformCore (const FormulaT &_constraint)`  
*The inverse of informing about a constraint.*
- virtual bool `addCore (ModuleInput::const_iterator formula)`  
*The module has to take the given sub-formula of the received formula into account.*
- virtual `Answer checkCore ()`  
*Checks the received formula for consistency.*
- virtual void `removeCore (ModuleInput::const_iterator formula)`  
*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound () const`  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel () const`  
*Clears the assignment, if any was found.*
- void `clearModels () const`

- **void cleanModel () const**

*Clears all assignments, if any was found.*
- **ModuleInput::iterator passedFormulaBegin ()**

*Substitutes variable occurrences by its model value in the model values of other variables.*
- **ModuleInput::iterator passedFormulaEnd ()**
- **void addOrigin (ModuleInput::iterator \_formula, const FormulaT &\_origin)**

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- **const FormulaT & getOrigins (ModuleInput::const\_iterator \_formula) const**

*Gets the origins of the passed formula at the given position.*
- **void getOrigins (const FormulaT &\_formula, FormulasT &\_origins) const**
- **void getOrigins (const FormulaT &\_formula, FormulaSetT &\_origins) const**
- **std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator \_formula, const FormulaT &\_origin)**
- **std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator \_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)**
- **void informBackends (const FormulaT &\_constraint)**

*Informs all backends of this module about the given constraint.*
- **virtual void addConstraintToInform (const FormulaT &\_constraint)**

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- **std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const\_iterator \_subformula)**

*Copies the given sub-formula of the received formula to the passed formula.*
- **bool originInReceivedFormula (const FormulaT &\_origin) const**
- **std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &\_formula)**

*Adds the given formula to the passed formula with no origin.*
- **std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &\_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)**

*Adds the given formula to the passed formula.*
- **std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &\_formula, const FormulaT &\_origin)**

*Adds the given formula to the passed formula.*
- **void generateTrivialInfeasibleSubset ()**

*Stores the trivial infeasible subset being the set of received formulas.*
- **void receivedFormulasAsInfeasibleSubset (ModuleInput::const\_iterator \_subformula)**

*Stores an infeasible subset consisting only of the given received formula.*
- **std::vector< FormulaT >::const\_iterator findBestOrigin (const std::vector< FormulaT > &\_origins) const**
- **void getInfeasibleSubsets ()**

*Copies the infeasible subsets of the passed formula.*
- **std::vector< FormulaSetT > getInfeasibleSubsets (const Module &\_backend) const**

*Get the infeasible subsets the given backend provides.*
- **const Model & backendsModel () const**

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- **void getBackendsModel () const**
- **void getBackendsAllModels () const**

*Stores all models of a backend in the list of all models of this module.*
- **virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator \_subformula, bool \_ignoreOrigins=false)**

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- **void clearPassedFormula ()**
- **std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const**

*Merges the two vectors of sets into the first one.*
- **size\_t determine\_smallest\_origin (const std::vector< FormulaT > &origins) const**

- bool `probablyLooping` (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const  
*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< FormulaT > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)  
*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &&\_premise=std::vector< FormulaT >())
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, std::vector< FormulaT > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &\_premise=std::vector< FormulaT >())
- void `splitUnequalConstraint` (const FormulaT &)  
*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const FormulaT &formula)  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< FormulaT > & `getInformationRelevantFormulas` ()  
*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (LemmaLevel level)  
*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const Model &\_modelA, const Model &\_modelB)  
*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< FormulaSetT > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- Manager \*const `mpManager`  
*A reference to the manager.*
- Model `mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< Model > `mAllModels`  
*Stores all satisfying assignments.*
- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< TheoryPropagation > `mTheoryPropagations`
- std::atomic< Answer > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- **Conditionals mFoundAnswer**

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- std::vector< [Module \\*](#) > mUsedBackends

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- std::vector< [Module \\*](#) > mAllBackends

*The backends of this module which have been used.*

- std::vector< [Lemma](#) > mLemmas

*Stores the lemmas being valid formulas this module or its backends made.*

- [ModuleInput::iterator](#) mFirstSubformulaToPass

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- carl::FastSet< [FormulaT](#) > mConstraintsToInform

*Stores the constraints which the backends must be informed about.*

- carl::FastSet< [FormulaT](#) > mInformedConstraints

*Stores the position of the first constraint of which no backend has been informed about.*

- [ModuleInput::const\\_iterator](#) mFirstUncheckedReceivedSubformula

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- unsigned mSmallerMusesCheckCounter

*Counter used for the generation of the smt2 files to check for smaller muses.*

- std::vector< std::size\_t > mVariableCounters

*Maps variables to the number of their occurrences.*

### 0.15.367.1 Member Enumeration Documentation

#### 0.15.367.1.1 LemmaType [enum smtrat::Module::LemmaType : unsigned \[strong\], \[inherited\]](#)

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.367.2 Constructor & Destructor Documentation

#### 0.15.367.2.1 PModule() [smtrat::PModule::PModule \(](#)

```
 const ModuleInput * _formula,
 Conditionals & _foundAnswer,
 Manager * _manager = nullptr,
 std::string module_name = "PModule")
```

### 0.15.367.3 Member Function Documentation

#### 0.15.367.3.1 add() [bool smtrat::PModule::add \(](#)

```
 ModuleInput::const_iterator _subformula)
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.367.3.2 `addConstraintToInform()`** `void smtrat::Module::addConstraintToInform ( const FormulaT & _constraint )` [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.367.3.3 `addCore()`** `virtual bool smtrat::Module::addCore ( ModuleInput::const_iterator formula )` [inline], [protected], [virtual], [inherited]

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICEModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::GBModule< Settings >](#).

**0.15.367.3.4 `addInformationRelevantFormula()`** `void smtrat::Module::addInformationRelevantFormula ( const FormulaT & formula )` [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

```
0.15.367.3.5 addLemma() void smtrat::Module::addLemma (
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| _lemma            | The eduction/lemma to store.                                          |
| _lt               | The type of the lemma.                                                |
| _preferredFormula | Hint for the next decision, which formula should be assigned to true. |

```
0.15.367.3.6 addOrigin() void smtrat::Module::addOrigin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

```
0.15.367.3.7 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

```
0.15.367.3.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat<-
::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

---

**0.15.367.3.9 addSubformulaToPassedFormula()** [2/3] std::pair<ModuleInput::iterator,bool> smrat→  
::Module::addSubformulaToPassedFormula ( const FormulaT & \_formula,  
const FormulaT & \_origin ) [inline], [protected], [inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                 |
| _origin  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.367.3.10 addSubformulaToPassedFormula()** [3/3] std::pair<ModuleInput::iterator,bool>  
smrat::Module::addSubformulaToPassedFormula ( const FormulaT & \_formula,  
const std::shared\_ptr<std::vector<FormulaT>> & \_origins ) [inline], [protected],  
[inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                                         |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                               |
| _origins | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.367.3.11 allModels()** const std::vector<Model>& smrat::Module::allModels () const [inline],  
[inherited]

#### Returns

All satisfying assignments, if existent.

**0.15.367.3.12 anAnswerFound()** bool smrat::Module::anAnswerFound () const [inline], [protected],  
[inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

**0.15.367.3.13 answerFound()** const smtrat::Conditionals& smtrat::Module::answerFound ( ) const [inline], [inherited]

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.367.3.14 appliedPreprocessing()** bool smtrat::PModule::appliedPreprocessing ( ) const [inline]

**Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.367.3.15 backendsModel()** const Model & smtrat::Module::backendsModel ( ) const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.367.3.16 branchAt() [1/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & \_premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]

**0.15.367.3.17 branchAt() [2/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, std::vector< FormulaT > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.367.3.18 branchAt() [3/4]** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & \_premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]

**0.15.367.3.19 branchAt() [4/4]** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, std::vector< FormulaT > && \_premise, bool \_leftCaseWeak = true,

```
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

#### 0.15.367.3.20 `check()` [Answer](#) `smtrat::PModule::check(`

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

#### 0.15.367.3.21 `checkCore()` [Answer](#) `smtrat::Module::checkCore( )` [protected], [inherited]

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::SymmetryModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::SplitSOSModule< Settings >](#),

`smrat::SATModule< Settings >, smrat::PFEModule< Settings >, smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::MCBModule< Settings >, smrat::LVEModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettings1 >, smrat::IntEqModule< Settings >, smrat::IntBlastModule< Settings >, smrat::IncWidthModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, smrat::ICPModule< ICPSettings1 >, smrat::ICEModule< Settings >, smrat::GBPPModule< Settings >, smrat::GBModule< Settings >, smrat::FPPModule< Settings >, smrat::FouMoModule< Settings >, smrat::ESModule< Settings >, smrat::EQPreprocessingModule< Settings >, smrat::EQModule< Settings >, smrat::EMModule< Settings >, smrat::CurryModule< Settings >, smrat::CubeLIAModule< Settings >, smrat::CSplitModule< Settings >, smrat::CNFerModule, smrat::BVMODULE< Settings >, and smrat::BEModule< Settings >.`

#### 0.15.367.3.22 **checkInfSubsetForMinimality()** `void smrat::Module::checkInfSubsetForMinimality (`

```
 std::vector< FormulaSet >::const_iterator _infsSubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

##### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

#### 0.15.367.3.23 **checkModel()** `unsigned smrat::Module::checkModel ( ) const [protected], [inherited]`

##### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

#### 0.15.367.3.24 **cleanModel()** `void smrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

#### 0.15.367.3.25 **clearLemmas()** `void smrat::Module::clearLemmas ( ) [inline], [inherited]`

Deletes all yet found lemmas.

#### 0.15.367.3.26 **clearModel()** `void smrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

#### 0.15.367.3.27 **clearModels()** `void smrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.367.3.28 clearPassedFormula()** void smtrat::Module::clearPassedFormula () [protected],  
[inherited]

**0.15.367.3.29 collectOrigins() [1/2]** void smtrat::Module::collectOrigins (  
const FormulaT & \_formula,  
FormulaSetT & \_origins ) const [inherited]

**0.15.367.3.30 collectOrigins() [2/2]** void smtrat::Module::collectOrigins (  
const FormulaT & \_formula,  
FormulasT & \_origins ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|          |                                                                                                      |
|----------|------------------------------------------------------------------------------------------------------|
| _formula | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| _origins | The set in which to store the origins.                                                               |

**0.15.367.3.31 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ()  
[inherited]

**0.15.367.3.32 constraintsToInform()** const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]

#### Returns

The constraints which the backends must be informed about.

**0.15.367.3.33 currentlySatisfied()** virtual unsigned smtrat::Module::currentlySatisfied (  
const FormulaT & ) const [inline], [virtual], [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.367.3.34 currentlySatisfiedByBackend()** unsigned smtrat::Module::currentlySatisfiedByBackend (  
const FormulaT & \_formula ) const [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.367.3.35 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

```
0.15.367.3.36 deinformCore() virtual void smtrat::Module::deinformCore (
 const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

```
0.15.367.3.37 determine_smallest_origin() size_t smtrat::Module::determine_smallest_origin (
 const std::vector< FormulaT > & origins) const [protected], [inherited]
```

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

```
0.15.367.3.38 eraseSubformulaFromPassedFormula() ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.367.3.39 excludeNotReceivedVariablesFromModel()** void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]  
Excludes all variables from the current model, which do not occur in the received formula.

**0.15.367.3.40 findBestOrigin()** std::vector< [FormulaT](#) >::const\_iterator smtrat::Module::findBestOrigin ( const std::vector< [FormulaT](#) > & *\_origins* ) const [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

**0.15.367.3.41 firstSubformulaToPass()** [ModuleInput](#)::const\_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.367.3.42 firstUncheckedReceivedSubformula()** [ModuleInput](#)::const\_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.367.3.43 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const [FormulaT](#) & *\_splittingVariable* ) [inline], [static], [inherited]

**0.15.367.3.44 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.367.3.45 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]

Stores all models of a backend in the list of all models of this module.

**0.15.367.3.46 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.367.3.47 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.367.3.48 getInfeasibleSubsets()** [2/2] `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets ( const Module & _backend ) const [protected], [inherited]`  
Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.367.3.49 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.367.3.50 getModelEqualities()** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.367.3.51 getOrigins()** [1/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.367.3.52 getOrigins()** [2/3] `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.367.3.53 getOrigins()** [3/3] `const FormulaT& smrat::Module::getOrigins (`  
`ModuleInput::const_iterator _formula ) const` [inline], [protected], [inherited]  
Gets the origins of the passed formula at the given position.

**Parameters**

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.367.3.54 getReceivedFormulaSimplified()** `std::pair< bool, FormulaT > smrat::PModule::get<`  
`ReceivedFormulaSimplified ( ) [virtual]`

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smrat::Module](#).

**0.15.367.3.55 hasLemmas()** `bool smrat::Module::hasLemmas ( ) [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.367.3.56 hasValidInfeasibleSubset()** `bool smrat::Module::hasValidInfeasibleSubset ( ) const`  
[inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.367.3.57 id()** `std::size_t smrat::Module::id ( ) const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.367.3.58 infeasibleSubsets()** `const std::vector<FormulaSetT>& smrat::Module::infeasible<`  
`Subsets ( ) const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.367.3.59 inform()** `bool smrat::Module::inform (`  
`const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.367.3.60 `informBackends()`** `void smrat::Module::informBackends ( const FormulaT & _constraint ) [inline], [protected], [inherited]`  
Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.367.3.61 `informCore()`** `virtual bool smrat::Module::informCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`  
Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in `smrat::UnionFindModule< Settings >`, `smrat::PBPPModule< Settings >`, `smrat::PBGaussModule< Settings >`, `smrat::NRAILModule< Settings >`, `smrat::NewCoveringModule< Settings >`, `smrat::NewCADModule< Settings >`, `smrat::LRAModule< Settings >`, `smrat::LRAModule< LRASettingsICP >`, `smrat::LRAModule< LRASettings1 >`, `smrat::IntBlastModule< Settings >`, `smrat::FPPModule< Settings >`, `smrat::EQModule< Settings >`, `smrat::CurryModule< Settings >`, `smrat::BVMModule< Settings >`, `smrat::ICPModule< Settings >`, `smrat::ICPModule< ICPSettings4 >`, and `smrat::ICPModule< ICPSettings1 >`.

**0.15.367.3.62 `informedConstraints()`** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.367.3.63 `init()`** `void smrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >`, `smrat::PBGaussModule< Settings >`, `smrat::NRAILModule< Settings >`, `smrat::NewCoveringModule< Settings >`, `smrat::NewCADModule< Settings >`, `smrat::LRAModule< Settings >`, `smrat::LRAModule< LRASettingsICP >`, `smrat::LRAModule< LRASettings1 >`, `smrat::IntBlastModule< Settings >`, `smrat::FPPModule< Settings >`, `smrat::EQModule< Settings >`, `smrat::CurryModule< Settings >`, and `smrat::BVMModule< Settings >`.

**0.15.367.3.64 `is_minimizing()`** `bool smtrat::Module::is_minimizing () const [inline], [inherited]`

**0.15.367.3.65 `isLemmaLevel()`** `bool smtrat::Module::isLemmaLevel (`

`LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.367.3.66 `isPreprocessor()`** `bool smtrat::PModule::isPreprocessor () const [inline]`

Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.367.3.67 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas () const [inline], [inherited]`

Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.367.3.68 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (`

`const std::vector< FormulaT > & _vecSetA,`

`const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

Parameters

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

Returns

The vector being the two given vectors merged.

**0.15.367.3.69 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`

Returns

The assignment of the current satisfiable result, if existent.

**0.15.367.3.70 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (`

`const Model & _modelA,`

`const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.367.3.71 `moduleName()`** `virtual std::string smtrat::Module::moduleName( ) const [inline], [virtual], [inherited]`

**Returns**

The name of the given module type as name.

Reimplemented in `smtrat::VSModule< Settings >`, `smtrat::UnionFindModule< Settings >`, `smtrat::UFCegarModule< Settings >`, `smtrat::STropModule< Settings >`, `smtrat::SplitSOSModule< Settings >`, `smtrat::PBPPModule< Settings >`, `smtrat::PBGaussModule< Settings >`, `smtrat::NRAILModule< Settings >`, `smtrat::NewCADModule< Settings >`, `smtrat::LVEModule< Settings >`, `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, `smtrat::LRAModule< LRASettings1 >`, `smtrat::IntEqModule< Settings >`, `smtrat::IntBlastModule< Settings >`, `smtrat::IncWidthModule< Settings >`, `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, `smtrat::ICPModule< ICPSettings1 >`, `smtrat::GBModule< Settings >`, `smtrat::FouMoModule< Settings >`, `smtrat::EQPreprocessingModule< Settings >`, `smtrat::EQModule< Settings >`, `smtrat::CurryModule< Settings >`, `smtrat::CubeLIAModule< Settings >`, `smtrat::CSplitModule< Settings >`, `smtrat::BVMModule< Settings >`, and `smtrat::BEModule< Settings >`.

**0.15.367.3.72 `objective()`** `carl::Variable smtrat::Module::objective( ) const [inline], [inherited]`

**0.15.367.3.73 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.367.3.74 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.367.3.75 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.367.3.76 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.367.3.77 pReceivedFormula()** const `ModuleInput*` smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.367.3.78 print()** void smtrat::Module::print ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]  
Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.367.3.79 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & `_out` = `std::cout` ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.367.3.80 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.367.3.81 printModel()** void smtrat::Module::printModel ( std::ostream & `_out` = `std::cout` ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.367.3.82 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.367.3.83 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
 Prints the vector of the received formula.

#### Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.367.3.84 probablyLooping()** bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & *\_branchingPolynomial*, const Rational & *\_branchingValue* ) const [protected], [inherited]

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.367.3.85 receivedFormulaChecked()** void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.367.3.86 receivedFormulasAsInfeasibleSubset()** void smtrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const\_iterator *\_subformula* ) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

**0.15.367.3.87 receivedVariable()** bool smtrat::Module::receivedVariable ( carl::Variable::Arg *\_var* ) const [inline], [inherited]

**0.15.367.3.88 remove()** void smtrat::PModule::remove ( ModuleInput::const\_iterator *\_subformula* ) [virtual]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>_subformula</i> | The sub formula of the received formula to remove. |
|--------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.367.3.89 removeCore()** virtual void smrat::Module::removeCore (  
    ModuleInput::const\_iterator formula ) [inline], [protected], [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>formula</i> | The sub formula of the received formula to remove. |
|----------------|----------------------------------------------------|

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.367.3.90 removeOrigin()** std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigin (   
    ModuleInput::iterator \_formula,  
    const FormulaT & \_origin ) [inline], [protected], [inherited]

**0.15.367.3.91 removeOrigins()** std::pair<ModuleInput::iterator,bool> smrat::Module::removeOrigins (   
    ModuleInput::iterator \_formula,  
    const std::shared\_ptr< std::vector< FormulaT >> & \_origins ) [inline], [protected], [inherited]

**0.15.367.3.92 rPassedFormula()** const ModuleInput& smrat::Module::rPassedFormula ( ) const  
[inline], [inherited]

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.367.3.93 rReceivedFormula()** const ModuleInput& smrat::Module::rReceivedFormula ( ) const  
[inline], [inherited]

#### Returns

A reference to the formula passed to this module.

**0.15.367.3.94 runBackends() [1/2]** virtual Answer smrat::PModule::runBackends ( ) [inline], [virtual]

Reimplemented from [smrat::Module](#).

**0.15.367.3.95 runBackends()** [2/2] [Answer](#) smtrat::PModule::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [virtual]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.367.3.96 setId()** void smtrat::Module::setId (

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

#### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <code>id</code> | The id to set this module's id to. |
|-----------------|------------------------------------|

**0.15.367.3.97 setThreadPriority()** void smtrat::Module::setThreadPriority (

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.367.3.98 solverState()** [Answer](#) smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.367.3.99 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint (

```
 const FormulaT & _unequalConstraint) [protected], [inherited]
```

Adds the following lemmas for the given constraint p!=0.

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                             |
|---------------------------------|---------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol !=. |
|---------------------------------|---------------------------------------------|

**0.15.367.3.100 `threadPriority()`** `thread_priority` `smtrat::Module::threadPriority () const [inline], [inherited]`

**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.367.3.101 `updateAllModels()`** `void smtrat::Module::updateAllModels () [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in `smtrat::SATModule< Settings >`.

**0.15.367.3.102 `updateLemmas()`** `void smtrat::Module::updateLemmas () [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.367.3.103 `updateModel()`** `void smtrat::PModule::updateModel () const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from `smtrat::Module`.

**0.15.367.3.104 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends () const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.367.4 Field Documentation

**0.15.367.4.1 `mAllBackends`** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.367.4.2 `mAllModels`** `std::vector<Model> smtrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

**0.15.367.4.3 `mBackendsFoundAnswer`** `std::atomic_bool* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.367.4.4 mConstraintsToInform** `carl::FastSet<FormulaT> smrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.367.4.5 mFinalCheck** `bool smrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.367.4.6 mFirstPosInLastBranches** `std::size_t smrat::Module::mFirstPosInLastBranches = 0` [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.367.4.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.367.4.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.367.4.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.367.4.10 mFullCheck** `bool smrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.367.4.11 mInfeasibleSubsets** `std::vector<FormulaSet> smrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.367.4.12 mInformedConstraints** `carl::FastSet<FormulaT> smrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.367.4.13 mLastBranches** `std::vector<Branching> smrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.367.4.14 mLemmas** `std::vector<Lemma> smrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.367.4.15 mModel** `Model smrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.367.4.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.367.4.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.367.4.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.367.4.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.367.4.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.367.4.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.367.4.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.367.4.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.367.4.24 mUsedBackends** std::vector<Module\*> smtrat::Module::mUsedBackends [protected], [inherited]  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.367.4.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]  
Maps variables to the number of their occurrences.

## 0.15.368 smtrat::cadcells::operators::properties::poly\_additional\_root\_outside Struct Reference

```
#include <properties.h>
```

### Public Member Functions

- size\_t **level** () const
- std::size\_t **hash\_on\_level** () const

**Data Fields**

- `datastructures::PolyRef poly`

**Static Public Attributes**

- static constexpr bool `is_flag` = false

**0.15.368.1 Member Function Documentation**

**0.15.368.1.1 `hash_on_level()`** std::size\_t smtrat::cadcells::operators::properties::poly\_additional\_root\_outside::hash\_on\_level () const [inline]

**0.15.368.1.2 `level()`** size\_t smtrat::cadcells::operators::properties::poly\_additional\_root\_outside::level () const [inline]

**0.15.368.2 Field Documentation**

**0.15.368.2.1 `is_flag`** constexpr bool smtrat::cadcells::operators::properties::poly\_additional\_root\_outside::is\_flag = false [static], [constexpr]

**0.15.368.2.2 `poly`** `datastructures::PolyRef` smtrat::cadcells::operators::properties::poly\_additional\_root\_outside::poly

**0.15.369 smtrat::cadcells::operators::properties::poly\_irreducible\_semi\_sgn\_inv Struct Reference**

```
#include <properties.h>
```

**Public Member Functions**

- size\_t `level()` const
- std::size\_t `hash_on_level()` const

**Data Fields**

- `datastructures::PolyRef poly`

**Static Public Attributes**

- static constexpr bool `is_flag` = false

**0.15.369.1 Member Function Documentation**

**0.15.369.1.1 `hash_on_level()`** std::size\_t smtrat::cadcells::operators::properties::poly\_irreducible\_semi\_sgn\_inv::hash\_on\_level () const [inline]

**0.15.369.1.2 `level()`** size\_t smtrat::cadcells::operators::properties::poly\_irreducible\_semi\_sgn\_inv::level () const [inline]

### 0.15.369.2 Field Documentation

**0.15.369.2.1 `is_flag`** constexpr bool smtrat::cadcells::operators::properties::poly\_irreducible\_<semi\_sgn\_inv::is\_flag = false [static], [constexpr]

**0.15.369.2.2 `poly`** [datastructures::PolyRef](#) smtrat::cadcells::operators::properties::poly\_<irreducible\_semi\_sgn\_inv::poly

## 0.15.370 **smtrat::cadcells::operators::properties::poly\_irreducible\_sgn\_inv Struct Reference**

```
#include <properties.h>
```

### Public Member Functions

- `size_t level()` const
- `std::size_t hash_on_level()` const

### Data Fields

- [datastructures::PolyRef poly](#)

### Static Public Attributes

- static constexpr bool `is_flag` = false

### 0.15.370.1 Member Function Documentation

**0.15.370.1.1 `hash_on_level()`** std::size\_t smtrat::cadcells::operators::properties::poly\_irreducible\_<sgn\_inv::hash\_on\_level() const [inline]

**0.15.370.1.2 `level()`** size\_t smtrat::cadcells::operators::properties::poly\_irreducible\_sgn\_inv\_<::level() const [inline]

### 0.15.370.2 Field Documentation

**0.15.370.2.1 `is_flag`** constexpr bool smtrat::cadcells::operators::properties::poly\_irreducible\_<sgn\_inv::is\_flag = false [static], [constexpr]

**0.15.370.2.2 `poly`** [datastructures::PolyRef](#) smtrat::cadcells::operators::properties::poly\_<irreducible\_sgn\_inv::poly

## 0.15.371 **smtrat::cadcells::operators::properties::poly\_ord\_inv Struct Reference**

```
#include <properties.h>
```

**Public Member Functions**

- `size_t level () const`
- `std::size_t hash_on_level () const`

**Data Fields**

- `datastructures::PolyRef poly`

**Static Public Attributes**

- `static constexpr bool is_flag = false`

**0.15.371.1 Member Function Documentation**

**0.15.371.1.1 hash\_on\_level()** `std::size_t smtrat::cadcells::operators::properties::poly_ord_inv::hash_on_level () const [inline]`

**0.15.371.1.2 level()** `size_t smtrat::cadcells::operators::properties::poly_ord_inv::level () const [inline]`

**0.15.371.2 Field Documentation**

**0.15.371.2.1 is\_flag** `constexpr bool smtrat::cadcells::operators::properties::poly_ord_inv::is_flag = false [static], [constexpr]`

**0.15.371.2.2 poly** `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_ord_inv::poly`

**0.15.372 smtrat::cadcells::operators::properties::poly\_ord\_inv\_base Struct Reference**

```
#include <properties.h>
```

**Public Member Functions**

- `size_t level () const`
- `std::size_t hash_on_level () const`

**Data Fields**

- `datastructures::PolyRef poly`

**Static Public Attributes**

- `static constexpr bool is_flag = false`

**0.15.372.1 Member Function Documentation**

**0.15.372.1.1 hash\_on\_level()** `std::size_t smtrat::cadcells::operators::properties::poly_ord_inv_base::hash_on_level () const [inline]`

```
0.15.372.1.2 level() size_t smtrat::cadcells::operators::properties::poly_ord_inv_base::level () const [inline]
```

### 0.15.372.2 Field Documentation

```
0.15.372.2.1 is_flag constexpr bool smtrat::cadcells::operators::properties::poly_ord_inv_base::is_flag = false [static], [constexpr]
```

```
0.15.372.2.2 poly datastructures::PolyRef smtrat::cadcells::operators::properties::poly_ord_inv_base::poly
```

## 0.15.373 smtrat::cadcells::operators::properties::poly\_pdel Struct Reference

```
#include <properties.h>
```

### Public Member Functions

- `size_t level () const`
- `std::size_t hash_on_level () const`

### Data Fields

- `datastructures::PolyRef poly`

### Static Public Attributes

- `static constexpr bool is_flag = false`

### 0.15.373.1 Member Function Documentation

```
0.15.373.1.1 hash_on_level() std::size_t smtrat::cadcells::operators::properties::poly_pdel::hash_on_level () const [inline]
```

```
0.15.373.1.2 level() size_t smtrat::cadcells::operators::properties::poly_pdel::level () const [inline]
```

### 0.15.373.2 Field Documentation

```
0.15.373.2.1 is_flag constexpr bool smtrat::cadcells::operators::properties::poly_pdel::is_flag = false [static], [constexpr]
```

```
0.15.373.2.2 poly datastructures::PolyRef smtrat::cadcells::operators::properties::poly_pdel::poly
```

## 0.15.374 smtrat::cadcells::operators::properties::poly\_semi\_sgn\_inv Struct Reference

```
#include <properties.h>
```

**Public Member Functions**

- `size_t level () const`
- `std::size_t hash_on_level () const`

**Data Fields**

- `datastructures::PolyRef poly`

**Static Public Attributes**

- `static constexpr bool is_flag = false`

**0.15.374.1 Member Function Documentation**

**0.15.374.1.1 hash\_on\_level()** `std::size_t smtrat::cadcells::operators::properties::poly_semi_sgn_inv::hash_on_level () const [inline]`

**0.15.374.1.2 level()** `size_t smtrat::cadcells::operators::properties::poly_semi_sgn_inv::level () const [inline]`

**0.15.374.2 Field Documentation**

**0.15.374.2.1 is\_flag** `constexpr bool smtrat::cadcells::operators::properties::poly_semi_sgn_inv::is_flag = false [static], [constexpr]`

**0.15.374.2.2 poly** `datastructures::PolyRef smtrat::cadcells::operators::properties::poly_semi_sgn_inv::poly`

**0.15.375 smtrat::cadcells::operators::properties::poly\_sgn\_inv Struct Reference**

```
#include <properties.h>
```

**Public Member Functions**

- `size_t level () const`
- `std::size_t hash_on_level () const`

**Data Fields**

- `datastructures::PolyRef poly`

**Static Public Attributes**

- `static constexpr bool is_flag = false`

**0.15.375.1 Member Function Documentation**

**0.15.375.1.1 hash\_on\_level()** `std::size_t smtrat::cadcells::operators::properties::poly_sgn_inv::hash_on_level () const [inline]`

```
0.15.375.1.2 level() size_t smtrat::cadcells::operators::properties::poly_sgn_inv::level ()
const [inline]
```

## 0.15.375.2 Field Documentation

```
0.15.375.2.1 is_flag constexpr bool smtrat::cadcells::operators::properties::poly_sgn_inv::is_flag = false [static], [constexpr]
```

```
0.15.375.2.2 poly datastructures::PolyRef smtrat::cadcells::operators::properties::poly_sgn_inv::poly
```

## 0.15.376 smtrat::cadcells::representation::util::PolyDelineation Struct Reference

```
#include <util.h>
```

### Data Fields

- boost::container::flat\_set< std::size\_t > [delineated\\_roots](#)
- std::size\_t [critical\\_lower\\_root](#) = 0
- std::size\_t [critical\\_upper\\_root](#) = 0

## 0.15.376.1 Field Documentation

```
0.15.376.1.1 critical_lower_root std::size_t smtrat::cadcells::representation::util::PolyDelineation::critical_lower_root = 0
```

```
0.15.376.1.2 critical_upper_root std::size_t smtrat::cadcells::representation::util::PolyDelineation::critical_upper_root = 0
```

```
0.15.376.1.3 delineated_roots boost::container::flat_set<std::size_t> smtrat::cadcells::representation::util::PolyDelineation::delineated_roots
```

## 0.15.377 smtrat::cadcells::representation::util::PolyDelineations Struct Reference

```
#include <util.h>
```

### Public Member Functions

- auto & [get](#) (const [datastructures::PolyRef](#) &poly)

### Data Fields

- boost::container::flat\_map< [datastructures::PolyRef](#), [PolyDelineation](#) > [data](#)

## 0.15.377.1 Member Function Documentation

```
0.15.377.1.1 get() auto& smtrat::cadcells::representation::util::PolyDelineations::get (
 const datastructures::PolyRef & poly) [inline]
```

### 0.15.377.2 Field Documentation

**0.15.377.2.1 data** boost::container::flat\_map<[datastructures::PolyRef](#),[PolyDelineation](#)> smtrat<->  
::cadcells::representation::util::PolyDelineations::data

## 0.15.378 smtrat::cad::ProjectionPolynomialInformation::PolyInfo Struct Reference

```
#include <ProjectionInformation.h>
```

### Data Fields

- [Origin origin](#)

*Origins of this polynomial.*

### 0.15.378.1 Field Documentation

**0.15.378.1.1 origin** [Origin](#) smtrat::cad::ProjectionPolynomialInformation::PolyInfo::origin  
Origins of this polynomial.

## 0.15.379 smtrat::cad::PolynomialComparator< PolynomialGetter > Struct Template Reference

```
#include <PolynomialLiftingQueue.h>
```

### Public Member Functions

- [PolynomialComparator](#) (const PolynomialGetter \*pg, std::size\_t level)
- bool [operator\(\)](#) (std::size\_t lhs, std::size\_t rhs) const

### 0.15.379.1 Constructor & Destructor Documentation

**0.15.379.1.1 PolynomialComparator()** template<typename PolynomialGetter >  
smtrat::cad::PolynomialComparator< PolynomialGetter >::PolynomialComparator (

```
 const PolynomialGetter * pg,
 std::size_t level) [inline]
```

### 0.15.379.2 Member Function Documentation

**0.15.379.2.1 operator()** template<typename PolynomialGetter >  
bool smtrat::cad::PolynomialComparator< PolynomialGetter >::operator() (

```
 std::size_t lhs,
 std::size_t rhs) const [inline]
```

## 0.15.380 smtrat::cad::PolynomialLiftingQueue< PolynomialGetter > Class Template Reference

```
#include <PolynomialLiftingQueue.h>
```

## Public Member Functions

- `PolynomialLiftingQueue` (const `PolynomialGetter` \*`pg`, `std::size_t` `level`)
- auto `insert` (`std::size_t` `id`)
- auto `erase` (`std::size_t` `id`)
- auto `begin` () const
- auto `end` () const
- auto `size` () const
- void `disable` (`std::size_t` `id`)
- void `restore` (`std::size_t` `id`)

## Friends

- template<typename PG>  
`std::ostream & operator<<` (`std::ostream &os`, const `PolynomialLiftingQueue< PG >` &`plq`)

### 0.15.380.1 Constructor & Destructor Documentation

**0.15.380.1.1 `PolynomialLiftingQueue()`** template<typename `PolynomialGetter`>  
`smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >`::`PolynomialLiftingQueue` (  
    const `PolynomialGetter` \* `pg`,  
    `std::size_t` `level`) [inline]

### 0.15.380.2 Member Function Documentation

**0.15.380.2.1 `begin()`** template<typename `PolynomialGetter`>  
auto `smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >`::`begin` () const [inline]

**0.15.380.2.2 `disable()`** template<typename `PolynomialGetter`>  
void `smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >`::`disable` (  
    `std::size_t` `id`) [inline]

**0.15.380.2.3 `end()`** template<typename `PolynomialGetter`>  
auto `smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >`::`end` () const [inline]

**0.15.380.2.4 `erase()`** template<typename `PolynomialGetter`>  
auto `smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >`::`erase` (  
    `std::size_t` `id`) [inline]

**0.15.380.2.5 `insert()`** template<typename `PolynomialGetter`>  
auto `smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >`::`insert` (  
    `std::size_t` `id`) [inline]

**0.15.380.2.6 `restore()`** template<typename `PolynomialGetter`>  
void `smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >`::`restore` (  
    `std::size_t` `id`) [inline]

```
0.15.380.2.7 size() template<typename PolynomialGetter >
auto smtrat::cad::PolynomialLiftingQueue< PolynomialGetter >::size () const [inline]
```

### 0.15.380.3 Friends And Related Function Documentation

```
0.15.380.3.1 operator<< template<typename PolynomialGetter >
template<typename PG >
std::ostream& operator<< (
 std::ostream & os,
 const PolynomialLiftingQueue< PG > & plq) [friend]
```

## 0.15.381 smtrat::cadcells::datastructures::PolyPool Class Reference

A pool for polynomials.

```
#include <polynomials.h>
```

### Public Member Functions

- **PolyPool** (const Polynomial::ContextType &context)  
*Constructs a polynomial pool with respect to a variable ordering.*
- const **VariableOrdering & var\_order** () const
- **PolyRef insert** (const **Polynomial &poly**)
- **PolyRef operator()** (const **Polynomial &poly**)
- const **Polynomial & get** (const **PolyRef &ref**) const
- const **Polynomial & operator()** (const **PolyRef &ref**) const
- bool **known** (const **Polynomial &poly**) const
- void **clear\_levels** (size\_t level)
- const **Polynomial::ContextType & get\_context** () const

### Friends

- std::ostream & **operator<<** (std::ostream &os, const **PolyPool &data**)

### 0.15.381.1 Detailed Description

A pool for polynomials.

The polynomials are stored in a table, that is, a list of lists of polynomials of a given level.

### 0.15.381.2 Constructor & Destructor Documentation

```
0.15.381.2.1 PolyPool() smtrat::cadcells::datastructures::PolyPool::PolyPool (
 const Polynomial::ContextType & context) [inline]
```

Constructs a polynomial pool with respect to a variable ordering.

#### Parameters

|                  |                                                      |
|------------------|------------------------------------------------------|
| <i>var_order</i> | The variable ordering determining polynomial levels. |
|------------------|------------------------------------------------------|

### 0.15.381.3 Member Function Documentation

**0.15.381.3.1 `clear_levels()`** void smtrat::cadcells::datastructures::PolyPool::clear\_levels ( size\_t level ) [inline]

**0.15.381.3.2 `get()`** const Polynomial& smtrat::cadcells::datastructures::PolyPool::get ( const PolyRef & ref ) const [inline]

**0.15.381.3.3 `get_context()`** const Polynomial::ContextType& smtrat::cadcells::datastructures::PolyPool::get\_context ( ) const [inline]

**0.15.381.3.4 `insert()`** PolyRef smtrat::cadcells::datastructures::PolyPool::insert ( const Polynomial & poly ) [inline]

**0.15.381.3.5 `known()`** bool smtrat::cadcells::datastructures::PolyPool::known ( const Polynomial & poly ) const [inline]

**0.15.381.3.6 `operator()()` [1/2]** PolyRef smtrat::cadcells::datastructures::PolyPool::operator() ( const Polynomial & poly ) [inline]

**0.15.381.3.7 `operator()()` [2/2]** const Polynomial& smtrat::cadcells::datastructures::PolyPool::operator() ( const PolyRef & ref ) const [inline]

**0.15.381.3.8 `var_order()`** const VariableOrdering& smtrat::cadcells::datastructures::PolyPool::var\_order ( ) const [inline]

#### 0.15.381.4 Friends And Related Function Documentation

**0.15.381.4.1 `operator<<`** std::ostream& operator<< ( std::ostream & os, const PolyPool & data ) [friend]

### 0.15.382 smtrat::cadcells::datastructures::detail::PolyProperties Struct Reference

```
#include <projections.h>
```

#### Data Fields

- std::map< PolyRef, PolyRef > res
- std::optional< PolyRef > disc
- std::optional< PolyRef > ldcf
- std::vector< PolyRef > factors\_nonconst

#### 0.15.382.1 Field Documentation

**0.15.382.1.1 disc** std::optional<[PolyRef](#)> smtrat::cadcells::datastructures::detail::PolyProperties::disc

**0.15.382.1.2 factors\_nonconst** std::vector<[PolyRef](#)> smtrat::cadcells::datastructures::detail::PolyProperties::factors\_nonconst

**0.15.382.1.3 ldcf** std::optional<[PolyRef](#)> smtrat::cadcells::datastructures::detail::PolyProperties::ldcf

**0.15.382.1.4 res** std::map<[PolyRef](#), [PolyRef](#)> smtrat::cadcells::datastructures::detail::PolyProperties::res

## 0.15.383 smtrat::cadcells::datastructures::PolyRef Struct Reference

Refers to a polynomial.

```
#include <polynomials.h>
```

### Data Fields

- size\_t **level**  
*The level of the polynomial.*
- size\_t **id**  
*The id of the polynomial with respect to its level.*

### 0.15.383.1 Detailed Description

Refers to a polynomial.

### 0.15.383.2 Field Documentation

**0.15.383.2.1 id** size\_t smtrat::cadcells::datastructures::PolyRef::id  
The id of the polynomial with respect to its level.

**0.15.383.2.2 level** size\_t smtrat::cadcells::datastructures::PolyRef::level  
The level of the polynomial.

## 0.15.384 smtrat::PolyTree Class Reference

```
#include <PolyTree.h>
```

### Public Types

- enum class **Type** : unsigned { [VARIABLE](#) , [CONSTANT](#) , [SUM](#) , [PRODUCT](#) }

### Public Member Functions

- [PolyTree](#) (const [Poly](#) &\_poly)
- const [PolyTree](#) & [left](#) () const
- const [PolyTree](#) & [right](#) () const
- [carl::Variable::Arg](#) [variable](#) () const
- const [Integer](#) & [constant](#) () const

- `Type type () const`
- `const Poly & poly () const`

### 0.15.384.1 Member Enumeration Documentation

#### 0.15.384.1.1 Type `enum smtrat::PolyTree::Type : unsigned [strong]`

Enumerator

|          |
|----------|
| VARIABLE |
| CONSTANT |
| SUM      |
| PRODUCT  |

### 0.15.384.2 Constructor & Destructor Documentation

#### 0.15.384.2.1 `PolyTree()` `smtrat::PolyTree::PolyTree (const Poly & _poly )`

### 0.15.384.3 Member Function Documentation

#### 0.15.384.3.1 `constant()` `const Integer & smtrat::PolyTree::constant () const`

#### 0.15.384.3.2 `left()` `const PolyTree & smtrat::PolyTree::left () const`

#### 0.15.384.3.3 `poly()` `const Poly & smtrat::PolyTree::poly () const`

#### 0.15.384.3.4 `right()` `const PolyTree & smtrat::PolyTree::right () const`

#### 0.15.384.3.5 `type()` `PolyTree::Type smtrat::PolyTree::type () const`

#### 0.15.384.3.6 `variable()` `carl::Variable::Arg smtrat::PolyTree::variable () const`

### 0.15.385 smtrat::PolyTreeContent Class Reference

#include <PolyTree.h>

#### Public Member Functions

- `PolyTreeContent (const Poly & _poly, PolyTree::Type _type, const PolyTree & _left, const PolyTree & _right)`
- `PolyTreeContent (carl::Variable::Arg _variable)`
- `PolyTreeContent (Integer _constant)`
- `~PolyTreeContent ()`
- `const Poly & poly () const`
- `bool operator== (const PolyTreeContent & _other) const`

**Friends**

- class [PolyTree](#)

**0.15.385.1 Constructor & Destructor Documentation**

**0.15.385.1.1 PolyTreeContent() [1/3]** smtrat::PolyTreeContent::PolyTreeContent (

```
const Poly & _poly,
PolyTree::Type _type,
const PolyTree & _left,
const PolyTree & _right) [inline]
```

**0.15.385.1.2 PolyTreeContent() [2/3]** smtrat::PolyTreeContent::PolyTreeContent (

```
carl::Variable::Arg _variable) [inline]
```

**0.15.385.1.3 PolyTreeContent() [3/3]** smtrat::PolyTreeContent::PolyTreeContent (

```
Integer _constant) [inline]
```

**0.15.385.1.4 ~PolyTreeContent()** smtrat::PolyTreeContent::~PolyTreeContent ( ) [inline]

**0.15.385.2 Member Function Documentation**

**0.15.385.2.1 operator==()** bool smtrat::PolyTreeContent::operator== (

```
const PolyTreeContent & _other) const [inline]
```

**0.15.385.2.2 poly()** const Poly& smtrat::PolyTreeContent::poly ( ) const [inline]

**0.15.385.3 Friends And Related Function Documentation**

**0.15.385.3.1 PolyTree** friend class [PolyTree](#) [friend]

**0.15.385.4 Field Documentation**

**0.15.385.4.1 mConstant** Integer smtrat::PolyTreeContent::mConstant

**0.15.385.4.2 mVariable** carl::Variable smtrat::PolyTreeContent::mVariable

**0.15.386 smtrat::PolyTreePool Class Reference**

```
#include <PolyTreePool.h>
```

**Public Member Functions**

- const PolyTreeContent \* get (const Poly &\_pol)

## Protected Member Functions

- `PolyTreePool ()`  
*Constructor of the pool.*
- `~PolyTreePool ()`

### 0.15.386.1 Constructor & Destructor Documentation

**0.15.386.1.1 PolyTreePool()** `smtrat::PolyTreePool::PolyTreePool () [inline], [protected]`  
Constructor of the pool.

**0.15.386.1.2 ~PolyTreePool()** `smtrat::PolyTreePool::~PolyTreePool () [inline], [protected]`

### 0.15.386.2 Member Function Documentation

**0.15.386.2.1 get()** `const PolyTreeContent * smtrat::PolyTreePool::get ( const Poly & _pol )`

## 0.15.387 smtrat::cad::Preprocessor Class Reference

#include <Preprocessor.h>

### Public Member Functions

- `Preprocessor (const std::vector< carl::Variable > &vars)`
- `void clear ()`
- `void addConstraint (const ConstraintT &c)`
- `void removeConstraint (const ConstraintT &c)`
- `bool preprocess ()`
- `void postprocessConflict (std::set< FormulaT > &mis) const`
- `const Model & model () const`
- `const auto & getConflict () const`
- `FormulaT simplify (const FormulaT &f) const`
- `template<typename Map > preprocessor::ConstraintUpdate result (const Map &oldC) const`

### Friends

- `std::ostream & operator<< (std::ostream &os, const Preprocessor &cadpp)`

### 0.15.387.1 Constructor & Destructor Documentation

**0.15.387.1.1 Preprocessor()** `smtrat::cad::Preprocessor::Preprocessor ( const std::vector< carl::Variable > & vars ) [inline]`

### 0.15.387.2 Member Function Documentation

**0.15.387.2.1 `addConstraint()`** void smtrat::cad::Preprocessor::addConstraint ( const ConstraintT & c )

**0.15.387.2.2 `clear()`** void smtrat::cad::Preprocessor::clear ( ) [inline]

**0.15.387.2.3 `getConflict()`** const auto& smtrat::cad::Preprocessor::getConflict ( ) const [inline]

**0.15.387.2.4 `model()`** const Model& smtrat::cad::Preprocessor::model ( ) const [inline]

**0.15.387.2.5 `postprocessConflict()`** void smtrat::cad::Preprocessor::postprocessConflict ( std::set< FormulaT > & mis ) const

**0.15.387.2.6 `preprocess()`** bool smtrat::cad::Preprocessor::preprocess ( )

**0.15.387.2.7 `removeConstraint()`** void smtrat::cad::Preprocessor::removeConstraint ( const ConstraintT & c )

**0.15.387.2.8 `result()`** template<typename Map > preprocessor::ConstraintUpdate smtrat::cad::Preprocessor::result ( const Map & oldC ) const [inline]

**0.15.387.2.9 `simplify()`** FormulaT smtrat::cad::Preprocessor::simplify ( const FormulaT & f ) const [inline]

### 0.15.387.3 Friends And Related Function Documentation

**0.15.387.3.1 `operator<<`** std::ostream& operator<< ( std::ostream & os, const Preprocessor & cadpp ) [friend]

## 0.15.388 smtrat::cad::PreprocessorSettings Struct Reference

#include <Preprocessor.h>

### Static Public Member Functions

- static void `register_settings` (SettingsParser &parser)
- static bool `register_hook` ()

### Data Fields

- bool `disable_variable_elimination` = false
- bool `disable_resultants` = false

### Static Public Attributes

- static const bool `dummy`

### 0.15.388.1 Member Function Documentation

**0.15.388.1.1 register\_hook()** static bool smtrat::cad::PreprocessorSettings::register\_hook ( )  
[inline], [static]

**0.15.388.1.2 register\_settings()** static void smtrat::cad::PreprocessorSettings::register\_settings  
(  
    SettingsParser & parser ) [inline], [static]

### 0.15.388.2 Field Documentation

**0.15.388.2.1 disable\_resultants** bool smtrat::cad::PreprocessorSettings::disable\_resultants = false

**0.15.388.2.2 disable\_variable\_elimination** bool smtrat::cad::PreprocessorSettings::disable\_variable\_elimination = false

**0.15.388.2.3 dummy** const bool smtrat::cad::PreprocessorSettings::dummy [static]

## 0.15.389 benchmax::settings::PresetSettings Struct Reference

```
#include <PresetSettings.h>
```

### Data Fields

- bool **rwth\_slurm**  
*Use slurm on the RWTH HPC cluster.*
- std::string **rwth\_slurm\_name**  
*Name for this job.*

### 0.15.389.1 Field Documentation

**0.15.389.1.1 rwth\_slurm** bool benchmax::settings::PresetSettings::rwth\_slurm  
Use slurm on the RWTH HPC cluster.

**0.15.389.1.2 rwth\_slurm\_name** std::string benchmax::settings::PresetSettings::rwth\_slurm\_name  
Name for this job.

## 0.15.390 smtrat::PriorityQueue< T, Compare > Class Template Reference

```
#include <DynamicPriorityQueue.h>
```

## Public Member Functions

- `PriorityQueue ()`
- `PriorityQueue (const Compare &comp)`
- `const auto & data () const`
- `auto & data ()`
- `auto find (const T &t) const`
- `auto begin () const`
- `auto begin ()`
- `auto end () const`
- `auto end ()`
- `auto erase (typename std::vector< T >::const_iterator it)`
- `auto erase (typename std::vector< T >::const_iterator it, typename std::vector< T >::const_iterator end)`
- `void fix ()`
- template<typename F >  
  `void removelf (F &&f)`
- `void clear ()`

## Data Fields

- `T elements`

*STL member.*

### 0.15.390.1 Constructor & Destructor Documentation

**0.15.390.1.1 `PriorityQueue()` [1/2]** template<typename T , typename Compare = std::less<T>>  
`smtrat::PriorityQueue< T, Compare >::PriorityQueue ( )` [inline], [explicit]

**0.15.390.1.2 `PriorityQueue()` [2/2]** template<typename T , typename Compare = std::less<T>>  
`smtrat::PriorityQueue< T, Compare >::PriorityQueue (`  
  `const Compare & comp )` [inline], [explicit]

### 0.15.390.2 Member Function Documentation

**0.15.390.2.1 `begin()` [1/2]** template<typename T , typename Compare = std::less<T>>  
`auto smtrat::PriorityQueue< T, Compare >::begin ( )` [inline]

**0.15.390.2.2 `begin()` [2/2]** template<typename T , typename Compare = std::less<T>>  
`auto smtrat::PriorityQueue< T, Compare >::begin ( ) const` [inline]

**0.15.390.2.3 `clear()`** template<typename T , typename Compare = std::less<T>>  
`void smtrat::PriorityQueue< T, Compare >::clear ( )` [inline]

**0.15.390.2.4 `data()` [1/2]** template<typename T , typename Compare = std::less<T>>  
`auto& smtrat::PriorityQueue< T, Compare >::data ( )` [inline]

**0.15.390.2.5 `data()` [2/2]** template<typename T , typename Compare = std::less<T>>  
const auto& smtrat::PriorityQueue< T, Compare >::data ( ) const [inline]

**0.15.390.2.6 `end()` [1/2]** template<typename T , typename Compare = std::less<T>>  
auto smtrat::PriorityQueue< T, Compare >::end ( ) [inline]

**0.15.390.2.7 `end()` [2/2]** template<typename T , typename Compare = std::less<T>>  
auto smtrat::PriorityQueue< T, Compare >::end ( ) const [inline]

**0.15.390.2.8 `erase()` [1/2]** template<typename T , typename Compare = std::less<T>>  
auto smtrat::PriorityQueue< T, Compare >::erase (  
 typename std::vector< T >::const\_iterator it ) [inline]

**0.15.390.2.9 `erase()` [2/2]** template<typename T , typename Compare = std::less<T>>  
auto smtrat::PriorityQueue< T, Compare >::erase (  
 typename std::vector< T >::const\_iterator it,  
 typename std::vector< T >::const\_iterator end ) [inline]

**0.15.390.2.10 `find()`** template<typename T , typename Compare = std::less<T>>  
auto smtrat::PriorityQueue< T, Compare >::find (  
 const T & t ) const [inline]

**0.15.390.2.11 `fix()`** template<typename T , typename Compare = std::less<T>>  
void smtrat::PriorityQueue< T, Compare >::fix ( ) [inline]

**0.15.390.2.12 `removelf()`** template<typename T , typename Compare = std::less<T>>  
template<typename F >  
void smtrat::PriorityQueue< T, Compare >::removeIf (  
 F && f ) [inline]

### 0.15.390.3 Field Documentation

**0.15.390.3.1 `elements`** T std::priority\_queue< T >::elements [inherited]  
STL member.

## 0.15.391 delta::Producer Class Reference

This class iteratively applies the operators to a smtlib file until no further simplifications can be performed.  
`#include <Producer.h>`

### Public Member Functions

- `Producer` (const `Checker` &checker, const `Settings` &settings)  
*Create a new manager that uses the given checker.*
- `void interrupt () const`
- `std::size_t operator() (Node &root)`  
*Apply simplifications to the given node.*

### 0.15.391.1 Detailed Description

This class iteratively applies the operators to a smtlib file until no further simplifications can be performed.

### 0.15.391.2 Constructor & Destructor Documentation

```
0.15.391.2.1 Producer() delta::Producer::Producer (
 const Checker & checker,
 const Settings & settings) [inline]
```

Create a new manager that uses the given checker.

#### Parameters

|                       |                                             |
|-----------------------|---------------------------------------------|
| <code>checker</code>  | <a href="#">Checker</a> to call the solver. |
| <code>settings</code> | <a href="#">Settings</a> object.            |

### 0.15.391.3 Member Function Documentation

```
0.15.391.3.1 interrupt() void delta::Producer::interrupt () const [inline]
```

```
0.15.391.3.2 operator() std::size_t delta::Producer::operator() (
 Node & root) [inline]
```

Apply simplifications to the given node.

#### Parameters

|                   |                                   |
|-------------------|-----------------------------------|
| <code>root</code> | <a href="#">Node</a> to simplify. |
|-------------------|-----------------------------------|

## 0.15.392 delta::ProgressBar Class Reference

This class provides a simple way to show progress bars on the command line.

```
#include <utils.h>
```

### Public Member Functions

- `void operator() ()`  
*Increase progress by one and show progress bar.*
- `void operator() (std::size_t steps)`  
*Increase progress by number of steps and show progress bar.*
- `void operator() (const std::pair< std::size_t, std::size_t > &p)`  
*Print a progress bar for a progress of progress / total.*
- `void operator() (std::size_t progress, std::size_t total)`  
*Print a progress bar for a progress of progress / total.*

### 0.15.392.1 Detailed Description

This class provides a simple way to show progress bars on the command line.

### 0.15.392.2 Member Function Documentation

**0.15.392.2.1 operator() [1/4]** void delta::ProgressBar::operator() () [inline]  
Increase progress by one and show progress bar.

**0.15.392.2.2 operator() [2/4]** void delta::ProgressBar::operator() (  
    const std::pair< std::size\_t, std::size\_t > & p ) [inline]  
Print a progress bar for a progress of progress / total.

#### Parameters

|          |           |
|----------|-----------|
| <i>p</i> | Progress. |
|----------|-----------|

**0.15.392.2.3 operator() [3/4]** void delta::ProgressBar::operator() (  
    std::size\_t progress,  
    std::size\_t total ) [inline]

Print a progress bar for a progress of progress / total.

#### Parameters

|                 |           |
|-----------------|-----------|
| <i>progress</i> | Progress. |
| <i>total</i>    | Total.    |

**0.15.392.2.4 operator() [4/4]** void delta::ProgressBar::operator() (  
    std::size\_t steps ) [inline]

Increase progress by number of steps and show progress bar.

#### Parameters

|              |                        |
|--------------|------------------------|
| <i>steps</i> | Steps to move forward. |
|--------------|------------------------|

## 0.15.393 smrat::cad::Projection< incrementality, backtracking, Settings > Class Template Reference

#include <Projection.h>

### Public Member Functions

- std::size\_t **dim** () const  
*Returns the dimension of the projection.*
- const auto & **vars** () const  
*Returns the variables used for projection.*
- void **reset** ()  
*Resets all datastructures, use the given variables from now on.*
- template<typename F>  
void **setRemoveCallback** (F &&f)  
*Sets a callback that is called whenever polynomials are removed.*

- carl::Bitset [addPolynomial](#) (const [Poly](#) &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual carl::Bitset [addPolynomial](#) (const [UPoly](#) &p, std::size\_t cid, bool isBound)=0
 

*Adds the given polynomial to the projection.*
- carl::Bitset [addEqConstraint](#) (const [Poly](#) &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual carl::Bitset [addEqConstraint](#) (const [UPoly](#) &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial of an equational constraint to the projection.*
- void [removePolynomial](#) (const [Poly](#) &p, std::size\_t cid, bool isBound)
 

*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual void [removePolynomial](#) (const [UPoly](#) &p, std::size\_t cid, bool isBound)=0
 

*Removes the given polynomial from the projection.*
- virtual std::size\_t [size](#) (std::size\_t level) const =0
 

*Get the size of the projection at the given level.*
- std::size\_t [size](#) () const
 

*Get the size of the projection.*
- virtual bool [empty](#) (std::size\_t level) const =0
 

*Check if the projection is empty at the given level.*
- virtual bool [empty](#) ()
 

*Check if the projection is empty.*
- [OptionalID](#) [getPolyForLifting](#) (std::size\_t level, [SampleLiftedWith](#) &slw)
 

*Get a polynomial from this level suited for lifting.*
- virtual bool [hasPolynomialByld](#) (std::size\_t level, std::size\_t id) const =0
 

*Check if a polynomial with the given id exists at the given level.*
- virtual const [UPoly](#) & [getPolynomialByld](#) (std::size\_t level, std::size\_t id) const =0
 

*Get a polynomial with the given id at the given level.*
- virtual [Origin](#) [getOrigin](#) (std::size\_t level, std::size\_t id) const
 

*Get the origin of a polynomial with the given id at the given level.*

## Protected Types

- using [Constraints](#) = [CADConstraints](#)< Settings::backtracking >

## Protected Member Functions

- void [callRemoveCallback](#) (std::size\_t level, const [SampleLiftedWith](#) &slw) const
 

*Returns a fresh polynomial id for the given level.*
- std::size\_t [getID](#) (std::size\_t level)
 

*Frees a currently used polynomial id for the given level.*
- carl::Variable [var](#) (std::size\_t level) const
 

*Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
- bool [canBePurgedByBounds](#) (const [UPoly](#) &p) const
 

*Checks whether a polynomial can safely be ignored due to the bounds.*
- bool [isPurged](#) (std::size\_t level, std::size\_t id)

## Protected Attributes

- const [Constraints](#) & [mConstraints](#)

*List of constraints used for projection.*
- std::vector< [PolynomialLiftingQueue](#)< BaseProjection > > [mLiftingQueues](#)

*List of lifting queues that can be used for incremental projection.*
- [ProjectionInformation](#) [mInfo](#)

*Additional info on projection, projection levels and projection polynomials.*
- [ProjectionOperator](#) [mOperator](#)

*The projection operator.*
- std::function< void(std::size\_t, const [SampleLiftedWith](#) &) > [mRemoveCallback](#)

*Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

### 0.15.393.1 Member Typedef Documentation

**0.15.393.1.1 Constraints** template<typename Settings >  
using smtrat::cad::BaseProjection< Settings >::Constraints = CADConstraints<Settings::backtracking>  
[protected], [inherited]

### 0.15.393.2 Member Function Documentation

**0.15.393.2.1 addEqConstraint() [1/2]** template<typename Settings >  
carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (  
 const Poly & p,  
 std::size\_t cid,  
 bool isBound ) [inline], [inherited]

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.393.2.2 addEqConstraint() [2/2]** template<typename Settings >  
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (  
 const UPoly & p,  
 std::size\_t cid,  
 bool isBound ) [inline], [virtual], [inherited]

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

**0.15.393.2.3 addPolynomial() [1/2]** template<typename Settings >  
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (  
 const Poly & p,  
 std::size\_t cid,  
 bool isBound ) [inline], [inherited]

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.393.2.4 addPolynomial() [2/2]** template<typename Settings >  
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (  
 const UPoly & p,  
 std::size\_t cid,  
 bool isBound ) [pure virtual], [inherited]

Adds the given polynomial to the projection.

Implemented in smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >, smtrat::cad::Projection< Incrementality::SIMPLE, Backtracking::HIDE, Settings >, smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >, and smtrat::cad::Projection< Incrementality::SIMPLE, Backtracking::SIMPLE, Settings >.

**0.15.393.2.5 callRemoveCallback()** template<typename Settings >  
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (  
 std::size\_t level,  
 const SampleLiftedWith & slw ) const [inline], [protected], [inherited]

**0.15.393.2.6 canBePurgedByBounds()** template<typename Settings >  
 bool [smtrat::cad::BaseProjection< Settings >](#)::canBePurgedByBounds ( const UPoly & p ) const [inline], [protected], [inherited]

Checks whether a polynomial can safely be ignored due to the bounds.

**0.15.393.2.7 dim()** template<typename Settings >  
 std::size\_t [smtrat::cad::BaseProjection< Settings >](#)::dim () const [inline], [inherited]

Returns the dimension of the projection.

**0.15.393.2.8 empty() [1/2]** template<typename Settings >  
 virtual bool [smtrat::cad::BaseProjection< Settings >](#)::empty () [inline], [virtual], [inherited]

**0.15.393.2.9 empty() [2/2]** template<typename Settings >  
 virtual bool [smtrat::cad::BaseProjection< Settings >](#)::empty ( std::size\_t level ) const [pure virtual], [inherited]

Implemented in [smtrat::qe::cad::Projection< Settings >](#), [smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED >](#)

[smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >](#), [smtrat::cad::ModelBasedProjection< Incrementality::NONE, BT, Settings >](#)

[smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#), and [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDDEN >](#)

**0.15.393.2.10 exportAsDot()** template<typename Settings >  
 virtual void [smtrat::cad::BaseProjection< Settings >](#)::exportAsDot ( std::ostream & ) const [inline], [virtual], [inherited]

Reimplemented in [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#), and [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDDEN >](#)

**0.15.393.2.11 freeID()** template<typename Settings >  
 void [smtrat::cad::BaseProjection< Settings >](#)::freeID ( std::size\_t level, std::size\_t id ) [inline], [protected], [inherited]

Frees a currently used polynomial id for the given level.

**0.15.393.2.12 getID()** template<typename Settings >  
 std::size\_t [smtrat::cad::BaseProjection< Settings >](#)::getID ( std::size\_t level ) [inline], [protected], [inherited]

Returns a fresh polynomial id for the given level.

**0.15.393.2.13 getOrigin()** template<typename Settings >  
 virtual [Origin smtrat::cad::BaseProjection< Settings >](#)::getOrigin ( std::size\_t level, std::size\_t id ) const [inline], [virtual], [inherited]

Reimplemented in [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#).

**0.15.393.2.14 getPolyForLifting()** template<typename Settings >  
 OptionalID [smtrat::cad::BaseProjection< Settings >](#)::getPolyForLifting ( std::size\_t level, SampleLiftedWith & slw ) [inline], [inherited]

Get a polynomial from this level suited for lifting.

```
0.15.393.2.15 getPolynomialById() template<typename Settings >
virtual const UPoly& smtrat::cad::BaseProjection< Settings >::getPolynomialById (
 std::size_t level,
 std::size_t id) const [pure virtual], [inherited]
```

Retrieves a polynomial from its id.

Implemented in `smtrat::qe::cad::Projection< Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

```
0.15.393.2.16 hasPolynomialById() template<typename Settings >
virtual bool smtrat::cad::BaseProjection< Settings >::hasPolynomialById (
 std::size_t level,
 std::size_t id) const [pure virtual], [inherited]
```

Implemented in `smtrat::qe::cad::Projection< Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

```
0.15.393.2.17 isPurged() template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::isPurged (
 std::size_t level,
 std::size_t id) [inline], [protected], [inherited]
```

```
0.15.393.2.18 removePolynomial() [1/2] template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.393.2.19 removePolynomial() [2/2] template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::removePolynomial (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [pure virtual], [inherited]
```

Removes the given polynomial from the projection.

Implemented in `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::NONE, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`, `smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::NONE, Backtracking::HIDE, Settings >`.

```
0.15.393.2.20 reset() template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::reset () [inline], [inherited]
```

Resets all datastructures, use the given variables from now on.

```
0.15.393.2.21 setRemoveCallback() template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
 F && f) [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

**0.15.393.2.22 size() [1/2]** template<typename Settings >  
`std::size_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]`

**0.15.393.2.23 size() [2/2]** template<typename Settings >  
`virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (`  
`std::size_t level ) const [pure virtual], [inherited]`

Implemented in `smtrat::qe::cad::Projection< Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::ModelBasedProjection< Incrementality::NONE, smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDDEN, Settings >`.

**0.15.393.2.24 var()** template<typename Settings >  
`carl::Variable smtrat::cad::BaseProjection< Settings >::var (`  
`std::size_t level ) const [inline], [protected], [inherited]`

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.393.2.25 vars()** template<typename Settings >  
`const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]`

Returns the variables used for projection.

### 0.15.393.3 Field Documentation

**0.15.393.3.1 mConstraints** template<typename Settings >  
`const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]`

**0.15.393.3.2 mInfo** template<typename Settings >  
`ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]`

Additional info on projection, projection levels and projection polynomials.

**0.15.393.3.3 mLiftingQueues** template<typename Settings >  
`std::vector<PolynomialLiftingQueue<BaseProjection> > smtrat::cad::BaseProjection< Settings >::mLiftingQueues [protected], [inherited]`

List of lifting queues that can be used for incremental projection.

**0.15.393.3.4 mOperator** template<typename Settings >  
`ProjectionOperator smtrat::cad::BaseProjection< Settings >::mOperator [protected], [inherited]`

The projection operator.

**0.15.393.3.5 mRemoveCallback** template<typename Settings >  
`std::function<void(std::size_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]`

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

## 0.15.394 smtrat::qe::cad::Projection< Settings > Class Template Reference

```
#include <Projection.h>
```

## Public Member Functions

- `Projection` (const `Constraints` &c)
- `void reset ()`
- `carl::Bitset addPolynomial` (const `UPoly` &p, std::size\_t cid, bool) override
- `void removePolynomial` (const `UPoly` &, std::size\_t cid, bool) override
- `bool testProjectionFactor` (std::size\_t level, std::size\_t id) const
- `void removeProjectionFactor` (std::size\_t level, std::size\_t id)
- `std::size_t size` (std::size\_t level) const override
- `bool empty` (std::size\_t level) const override
- `bool hasPolynomialById` (std::size\_t level, std::size\_t id) const override
- `const UPoly & getPolynomialById` (std::size\_t level, std::size\_t id) const override
 

*Retrieves a polynomial from its id.*
- `std::size_t dim () const`

*Returns the dimension of the projection.*
- `virtual std::size_t size` (std::size\_t level) const=0
- `std::size_t size () const`
- `std::size_t dim () const`

*Returns the dimension of the projection.*
- `const auto & vars () const`

*Returns the variables used for projection.*
- `template<typename F >`  
`void setRemoveCallback (F &&f)`

*Sets a callback that is called whenever polynomials are removed.*
- `carl::Bitset addPolynomial` (const `Poly` &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- `virtual carl::Bitset addPolynomial` (const `UPoly` &p, std::size\_t cid, bool isBound)=0
 

*Adds the given polynomial to the projection.*
- `carl::Bitset addEqConstraint` (const `Poly` &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- `virtual carl::Bitset addEqConstraint` (const `UPoly` &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial of an equational constraint to the projection.*
- `void removePolynomial` (const `Poly` &p, std::size\_t cid, bool isBound)
 

*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- `virtual void removePolynomial` (const `UPoly` &p, std::size\_t cid, bool isBound)=0
 

*Removes the given polynomial from the projection.*
- `std::size_t size () const`
- `virtual bool empty ()`
- `OptionalID getPolyForLifting` (std::size\_t level, SampleLiftedWith &slw)
 

*Get a polynomial from this level suited for lifting.*
- `virtual void exportAsDot` (std::ostream &) const
- `virtual Origin getOrigin` (std::size\_t level, std::size\_t id) const

## Protected Member Functions

- `void callRemoveCallback` (std::size\_t level, const SampleLiftedWith &slw) const
- `std::size_t getID` (std::size\_t level)
 

*Returns a fresh polynomial id for the given level.*
- `void freeID` (std::size\_t level, std::size\_t id)
 

*Frees a currently used polynomial id for the given level.*
- `carl::Variable var` (std::size\_t level) const
 

*Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
- `bool canBePurgedByBounds` (const `UPoly` &p) const
 

*Checks whether a polynomial can safely be ignored due to the bounds.*
- `bool isPurged` (std::size\_t level, std::size\_t id)

## Protected Attributes

- const `Constraints` & `mConstraints`
- `ProjectionInformation` `mInfo`  
*Additional info on projection, projection levels and projection polynomials.*
- `std::function< void(std::size_t, const SampleLiftedWith &) >` `mRemoveCallback`  
*Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

## Friends

- `template<typename S> std::ostream & operator<< (std::ostream &os, const Projection< S > &p)`

### 0.15.394.1 Constructor & Destructor Documentation

**0.15.394.1.1 `Projection()`** `template<typename Settings> smrat::qe::cad::Projection< Settings >::Projection( const Constraints & c ) [inline]`

### 0.15.394.2 Member Function Documentation

**0.15.394.2.1 `addEqConstraint()` [1/2]** `template<typename Settings> carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint( const Poly & p, std::size_t cid, bool isBound ) [inline], [inherited]`

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.394.2.2 `addEqConstraint()` [2/2]** `template<typename Settings> virtual carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint( const UPoly & p, std::size_t cid, bool isBound ) [inline], [virtual], [inherited]`

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in `smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

**0.15.394.2.3 `addPolynomial()` [1/3]** `template<typename Settings> carl::Bitset smrat::cad::BaseProjection< Settings >::addPolynomial( const Poly & p, std::size_t cid, bool isBound ) [inline], [inherited]`

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.394.2.4 `addPolynomial()` [2/3]** `template<typename Settings> virtual carl::Bitset smrat::cad::BaseProjection< Settings >::addPolynomial( const UPoly & p, std::size_t cid, bool isBound ) [pure virtual], [inherited]`

Adds the given polynomial to the projection.

Implemented in `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`, `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`, `smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`, and `smtrat::cad::Projection< Incrementality::SIMPLE, Backtracking::HIDE, Settings >`.

**0.15.394.2.5 addPolynomial() [3/3]** `template<typename Settings >`  
`carl::Bitset smtrat::qe::cad::Projection< Settings >::addPolynomial (`  
    `const UPoly & p,`  
    `std::size_t cid,`  
    `bool ) [inline], [override]`

**0.15.394.2.6 callRemoveCallback()** `template<typename Settings >`  
`void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (`  
    `std::size_t level,`  
    `const SampleLiftedWith & slw ) const [inline], [protected], [inherited]`

**0.15.394.2.7 canBePurgedByBounds()** `template<typename Settings >`  
`bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (`  
    `const UPoly & p ) const [inline], [protected], [inherited]`

Checks whether a polynomial can safely be ignored due to the bounds.

**0.15.394.2.8 dim() [1/2]** `template<typename Settings >`  
`std::size_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]`

Returns the dimension of the projection.

**0.15.394.2.9 dim() [2/2]** `template<typename Settings >`  
`std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]`

Returns the dimension of the projection.

**0.15.394.2.10 empty() [1/2]** `template<typename Settings >`  
`virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]`

**0.15.394.2.11 empty() [2/2]** `template<typename Settings >`  
`bool smtrat::qe::cad::Projection< Settings >::empty (`  
    `std::size_t level ) const [inline], [override], [virtual]`

Implements `smtrat::cad::BaseProjection< Settings >`.

**0.15.394.2.12 exportAsDot()** `template<typename Settings >`  
`virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (`  
    `std::ostream & ) const [inline], [virtual], [inherited]`

Reimplemented in `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`, and `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`.

**0.15.394.2.13 freeID()** `template<typename Settings >`  
`void smtrat::cad::BaseProjection< Settings >::freeID (`  
    `std::size_t level,`  
    `std::size_t id ) [inline], [protected], [inherited]`

Frees a currently used polynomial id for the given level.

**0.15.394.2.14 `getID()`** template<typename `Settings` >  
`std::size_t smtrat::cad::BaseProjection< Settings >::getID (`  
 `std::size_t level )` [inline], [protected], [inherited]  
 Returns a fresh polynomial id for the given level.

**0.15.394.2.15 `getOrigin()`** template<typename `Settings` >  
`virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin (`  
 `std::size_t level,`  
 `std::size_t id ) const` [inline], [virtual], [inherited]  
 Reimplemented in `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`.

**0.15.394.2.16 `getPolyForLifting()`** template<typename `Settings` >  
`OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (`  
 `std::size_t level,`  
 `SampleLiftedWith & slw )` [inline], [inherited]  
 Get a polynomial from this level suited for lifting.

**0.15.394.2.17 `getPolynomialById()`** template<typename `Settings` >  
`const UPoly& smtrat::qe::cad::Projection< Settings >::getPolynomialById (`  
 `std::size_t level,`  
 `std::size_t id ) const` [inline], [override], [virtual]  
 Retrieves a polynomial from its id.  
 Implements `smtrat::cad::BaseProjection< Settings >`.

**0.15.394.2.18 `hasPolynomialById()`** template<typename `Settings` >  
`bool smtrat::qe::cad::Projection< Settings >::hasPolynomialById (`  
 `std::size_t level,`  
 `std::size_t id ) const` [inline], [override], [virtual]  
 Implements `smtrat::cad::BaseProjection< Settings >`.

**0.15.394.2.19 `isPurged()`** template<typename `Settings` >  
`bool smtrat::cad::BaseProjection< Settings >::isPurged (`  
 `std::size_t level,`  
 `std::size_t id )` [inline], [protected], [inherited]

**0.15.394.2.20 `removePolynomial()` [1/3]** template<typename `Settings` >  
`void smtrat::cad::BaseProjection< Settings >::removePolynomial (`  
 `const Poly & p,`  
 `std::size_t cid,`  
 `bool isBound )` [inline], [inherited]

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.394.2.21 `removePolynomial()` [2/3]** template<typename `Settings` >  
`void smtrat::qe::cad::Projection< Settings >::removePolynomial (`  
 `const UPoly &,`  
 `std::size_t cid,`  
 `bool )` [inline], [override]

```
0.15.394.2.22 removePolynomial() [3/3] template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::removePolynomial (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [pure virtual], [inherited]
```

Removes the given polynomial from the projection.

Implemented in [smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >](#), [smtrat::cad::ModelBasedProjection< Incrementality::FULL, BT, Settings >](#), [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >](#), [smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >](#), and [smtrat::cad::Projection< Incrementality::NONE, Backtracking::HIDE, Settings >](#).

```
0.15.394.2.23 removeProjectionFactor() template<typename Settings >
void smtrat::qe::cad::Projection< Settings >::removeProjectionFactor (
 std::size_t level,
 std::size_t id) [inline]
```

```
0.15.394.2.24 reset() template<typename Settings >
void smtrat::qe::cad::Projection< Settings >::reset () [inline]
```

```
0.15.394.2.25 setRemoveCallback() template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
 F && f) [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

```
0.15.394.2.26 size() [1/4] template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size () const [inline], [inherited]
```

```
0.15.394.2.27 size() [2/4] template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size (
 void) [inline]
```

```
0.15.394.2.28 size() [3/4] template<typename Settings >
std::size_t smtrat::qe::cad::Projection< Settings >::size (
 std::size_t level) const [inline], [override], [virtual]
Implements smtrat::cad::BaseProjection< Settings >.
```

```
0.15.394.2.29 size() [4/4] template<typename Settings >
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
 void)
```

```
0.15.394.2.30 testProjectionFactor() template<typename Settings >
bool smtrat::qe::cad::Projection< Settings >::testProjectionFactor (
 std::size_t level,
 std::size_t id) const [inline]
```

**0.15.394.2.31 var()** template<typename Settings >  
 carl::Variable **smtrat::cad::BaseProjection< Settings >::var** (  
     std::size\_t level) const [inline], [protected], [inherited]  
 Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.394.2.32 vars()** template<typename Settings >  
 const auto& **smtrat::cad::BaseProjection< Settings >::vars** () const [inline], [inherited]  
 Returns the variables used for projection.

### 0.15.394.3 Friends And Related Function Documentation

**0.15.394.3.1 operator<<** template<typename Settings >  
 template<typename S >  
 std::ostream& **operator<<** (  
     std::ostream & os,  
     const **Projection< S >** & p ) [friend]

### 0.15.394.4 Field Documentation

**0.15.394.4.1 mConstraints** template<typename Settings >  
 const **Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints** [protected], [inherited]

**0.15.394.4.2 mInfo** template<typename Settings >  
 ProjectionInformation **smtrat::cad::BaseProjection< Settings >::mInfo** [protected], [inherited]  
 Additional info on projection, projection levels and projection polynomials.

**0.15.394.4.3 mRemoveCallback** template<typename Settings >  
 std::function<void(std::size\_t, const SampleLiftedWith&)> **smtrat::cad::BaseProjection< Settings >::mRemoveCallback** [protected], [inherited]  
 Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

## 0.15.395 **smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >** Class Template Reference

```
#include <Projection_EC.h>
```

### Public Member Functions

- **Projection** (const **Constraints &c**)
- void **reset** ()
- carl::Bitset **addPolynomial** (const **UPoly** &p, std::size\_t cid, bool isBound) override  
*Adds the given polynomial to the projection.*
- carl::Bitset **addEqConstraint** (const **UPoly** &p, std::size\_t cid, bool isBound) override  
*Adds the given polynomial of an equational constraint to the projection.*
- void **removePolynomial** (const **UPoly** &p, std::size\_t cid, bool isBound) override  
*Removes the given polynomial from the projection.*
- std::size\_t **size** (std::size\_t level) const override
- bool **empty** (std::size\_t level) const override

- carl::Bitset [projectNewPolynomial](#) (const [ConstraintSelection](#) &cs=carl::Bitset(true))
- bool [hasPolynomialById](#) (std::size\_t level, std::size\_t id) const override
- const [UPoly](#) & [getPolynomialById](#) (std::size\_t level, std::size\_t id) const override
  - Retrieves a polynomial from its id.*
- void [exportAsDot](#) (std::ostream &out) const override
- void [printPolynomialIDs](#) () const
- std::size\_t [dim](#) () const
  - Returns the dimension of the projection.*
- virtual std::size\_t [size](#) (std::size\_t level) const=0
- std::size\_t [size](#) () const
- std::size\_t [dim](#) () const
  - Returns the dimension of the projection.*
- const auto & [vars](#) () const
  - Returns the variables used for projection.*
- template<typename F>  
void [setRemoveCallback](#) (F &&f)
  - Sets a callback that is called whenever polynomials are removed.*
- carl::Bitset [addPolynomial](#) (const [Poly](#) &p, std::size\_t cid, bool isBound)
  - Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- carl::Bitset [addEqConstraint](#) (const [Poly](#) &p, std::size\_t cid, bool isBound)
  - Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- void [removePolynomial](#) (const [Poly](#) &p, std::size\_t cid, bool isBound)
  - Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- std::size\_t [size](#) () const
- virtual bool [empty](#) ()
- [OptionalID](#) [getPolyForLifting](#) (std::size\_t level, [SampleLiftedWith](#) &slw)
  - Get a polynomial from this level suited for lifting.*
- virtual [Origin](#) [getOrigin](#) (std::size\_t level, std::size\_t id) const

## Protected Member Functions

- void [callRemoveCallback](#) (std::size\_t level, const [SampleLiftedWith](#) &slw) const
- std::size\_t [getID](#) (std::size\_t level)
  - Returns a fresh polynomial id for the given level.*
- void [freeID](#) (std::size\_t level, std::size\_t id)
  - Frees a currently used polynomial id for the given level.*
- carl::Variable [var](#) (std::size\_t level) const
  - Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
- bool [canBePurgedByBounds](#) (const [UPoly](#) &p) const
  - Checks whether a polynomial can safely be ignored due to the bounds.*
- bool [isPurged](#) (std::size\_t level, std::size\_t id)

## Protected Attributes

- std::function< void(std::size\_t, const [SampleLiftedWith](#) &) > [mRemoveCallback](#)
  - Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

## Friends

- template<typename S>  
std::ostream & [operator<<](#) (std::ostream &os, const [Projection](#)< Incrementality::FULL, Backtracking::HIDE, S > &p)

### 0.15.395.1 Constructor & Destructor Documentation

**0.15.395.1.1 Projection()** template<typename Settings >  
`smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::Projection (`  
`const Constraints & c ) [inline]`

### 0.15.395.2 Member Function Documentation

**0.15.395.2.1 addEqConstraint() [1/2]** template<typename Settings >  
`carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint (`  
`const Poly & p,`  
`std::size_t cid,`  
`bool isBound ) [inline], [inherited]`

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.395.2.2 addEqConstraint() [2/2]** template<typename Settings >  
`carl::Bitset smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::addEqConstraint (`  
`const UPoly & p,`  
`std::size_t cid,`  
`bool isBound ) [inline], [override], [virtual]`

Adds the given polynomial of an equational constraint to the projection.

Polynomial already exists.

Reimplemented from `smtrat::cad::BaseProjection< Settings >`.

**0.15.395.2.3 addPolynomial() [1/2]** template<typename Settings >  
`carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (`  
`const Poly & p,`  
`std::size_t cid,`  
`bool isBound ) [inline], [inherited]`

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.395.2.4 addPolynomial() [2/2]** template<typename Settings >  
`carl::Bitset smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::addPolynomial (`  
`const UPoly & p,`  
`std::size_t cid,`  
`bool isBound ) [inline], [override], [virtual]`

Adds the given polynomial to the projection.

Implements `smtrat::cad::BaseProjection< Settings >`.

**0.15.395.2.5 callRemoveCallback()** template<typename Settings >  
`void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (`  
`std::size_t level,`  
`const SampleLiftedWith & slw ) const [inline], [protected], [inherited]`

**0.15.395.2.6 canBePurgedByBounds()** template<typename Settings >  
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds ( const UPoly & p ) const [inline], [protected], [inherited]  
Checks whether a polynomial can safely be ignored due to the bounds.

**0.15.395.2.7 dim() [1/2]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::dim () const [inline], [inherited]  
Returns the dimension of the projection.

**0.15.395.2.8 dim() [2/2]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::dim [inline]  
Returns the dimension of the projection.

**0.15.395.2.9 empty() [1/2]** template<typename Settings >  
virtual bool smtrat::cad::BaseProjection< Settings >::empty () [inline], [virtual], [inherited]

**0.15.395.2.10 empty() [2/2]** template<typename Settings >  
bool smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::empty ( std::size\_t level ) const [inline], [override], [virtual]  
Implements smtrat::cad::BaseProjection< Settings >.

**0.15.395.2.11 exportAsDot()** template<typename Settings >  
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::exportAsDot ( std::ostream & out ) const [inline], [override], [virtual]  
Reimplemented from smtrat::cad::BaseProjection< Settings >.

**0.15.395.2.12 freeID()** template<typename Settings >  
void smtrat::cad::BaseProjection< Settings >::freeID ( std::size\_t level, std::size\_t id ) [inline], [protected], [inherited]  
Frees a currently used polynomial id for the given level.

**0.15.395.2.13 getID()** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::getID ( std::size\_t level ) [inline], [protected], [inherited]  
Returns a fresh polynomial id for the given level.

**0.15.395.2.14 getOrigin()** template<typename Settings >  
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin ( std::size\_t level, std::size\_t id ) const [inline], [virtual], [inherited]  
Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

```
0.15.395.2.15 getPolyForLifting() template<typename Settings >
OptionalID smtrat::cad::BaseProjection< Settings >::getPolyForLifting (
 std::size_t level,
 SampleLiftedWith & slw) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

```
0.15.395.2.16 getPolynomialById() template<typename Settings >
const UPoly& smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::getPolynomialById (
 std::size_t level,
 std::size_t id) const [inline], [override], [virtual]
```

Retrieves a polynomial from its id.

Implements [smtrat::cad::BaseProjection< Settings >](#).

```
0.15.395.2.17 hasPolynomialById() template<typename Settings >
bool smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::hasPolynomialById (
 std::size_t level,
 std::size_t id) const [inline], [override], [virtual]
```

Implements [smtrat::cad::BaseProjection< Settings >](#).

```
0.15.395.2.18 isPurged() template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::isPurged (
 std::size_t level,
 std::size_t id) [inline], [protected], [inherited]
```

```
0.15.395.2.19 printPolynomialIDs() template<typename Settings >
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::printPolynomialIDs () const [inline]
```

```
0.15.395.2.20 projectNewPolynomial() template<typename Settings >
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::projectNewPolynomial (
 const ConstraintSelection & cs = carl::Bitset(true)) [inline]
```

```
0.15.395.2.21 removePolynomial() [1/2] template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.395.2.22 removePolynomial() [2/2] template<typename Settings >
void smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >::removePolynomial (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [inline], [override], [virtual]
```

Removes the given polynomial from the projection.

Implements [smtrat::cad::BaseProjection< Settings >](#).

**0.15.395.2.23 `reset()`** template<typename Settings >  
void [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >](#)::reset ( )  
[inline]

**0.15.395.2.24 `setRemoveCallback()`** template<typename Settings >  
template<typename F >  
void [smtrat::cad::BaseProjection< Settings >](#)::setRemoveCallback ( F && f ) [inline], [inherited]

Sets a callback that is called whenever polynomials are removed.

**0.15.395.2.25 `size() [1/4]`** template<typename Settings >  
std::size\_t [smtrat::cad::BaseProjection< Settings >](#)::size ( ) const [inline], [inherited]

**0.15.395.2.26 `size() [2/4]`** template<typename Settings >  
std::size\_t [smtrat::cad::BaseProjection< Settings >](#)::size ( void ) [inline]

**0.15.395.2.27 `size() [3/4]`** template<typename Settings >  
std::size\_t [smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >](#)::size ( std::size\_t level ) const [inline], [override], [virtual]  
Implements [smtrat::cad::BaseProjection< Settings >](#).

**0.15.395.2.28 `size() [4/4]`** template<typename Settings >  
virtual std::size\_t [smtrat::cad::BaseProjection< Settings >](#)::size ( void )

**0.15.395.2.29 `var()`** template<typename Settings >  
carl::Variable [smtrat::cad::BaseProjection< Settings >](#)::var ( std::size\_t level ) const [inline], [protected], [inherited]  
Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.395.2.30 `vars()`** template<typename Settings >  
const auto& [smtrat::cad::BaseProjection< Settings >](#)::vars ( ) const [inline], [inherited]  
Returns the variables used for projection.

### 0.15.395.3 Friends And Related Function Documentation

**0.15.395.3.1 `operator<<`** template<typename Settings >  
template<typename S >  
std::ostream& operator<< ( std::ostream & os, const [Projection< Incrementality::FULL, Backtracking::HIDE, S >](#) & p ) [friend]

### 0.15.395.4 Field Documentation

**0.15.395.4.1 mRemoveCallback** template<typename Settings >  
 std::function<void(std::size\_t, const SampleLiftedWith&) > smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]  
 Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

## 0.15.396 smtrat::cad::Projection< Incrementality::FULL, BT, Settings > Class Template Reference

```
#include <Projection_F.h>
```

### Public Member Functions

- **Projection** (const Constraints &c)
- void **reset** ()
- carl::Bitset **addPolynomial** (const UPoly &p, std::size\_t cid, bool isBound) override
 

*Adds the given polynomial to the projection.*
- void **removePolynomial** (const UPoly &p, std::size\_t cid, bool isBound) override
 

*Removes the given polynomial from the projection.*
- std::size\_t **size** (std::size\_t level) const override
- bool **empty** (std::size\_t level) const override
- carl::Bitset **projectNewPolynomial** (const ConstraintSelection &cs=carl::Bitset(true))
- bool **hasPolynomialById** (std::size\_t level, std::size\_t id) const override
- const UPoly & **getPolynomialById** (std::size\_t level, std::size\_t id) const override
 

*Retrieves a polynomial from its id.*
- void **exportAsDot** (std::ostream &out) const override
- Origin **getOrigin** (std::size\_t level, std::size\_t id) const override
- std::size\_t **dim** () const
 

*Returns the dimension of the projection.*
- virtual std::size\_t **size** (std::size\_t level) const=0
- std::size\_t **size** () const
- std::size\_t **dim** () const
 

*Returns the dimension of the projection.*
- const auto & **vars** () const
 

*Returns the variables used for projection.*
- template<typename F >
 void **setRemoveCallback** (F &&f)
 

*Sets a callback that is called whenever polynomials are removed.*
- carl::Bitset **addPolynomial** (const Poly &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- carl::Bitset **addEqConstraint** (const Poly &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual carl::Bitset **addEqConstraint** (const UPoly &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial of an equational constraint to the projection.*
- void **removePolynomial** (const Poly &p, std::size\_t cid, bool isBound)
 

*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- std::size\_t **size** () const
- virtual bool **empty** ()
- **OptionalID getPolyForLifting** (std::size\_t level, SampleLiftedWith &slw)
 

*Get a polynomial from this level suited for lifting.*

## Protected Member Functions

- void `callRemoveCallback` (std::size\_t level, const `SampleLiftedWith` &slw) const
  - std::size\_t `getID` (std::size\_t level)

*Returns a fresh polynomial id for the given level.*
  - void `freeID` (std::size\_t level, std::size\_t id)

*Frees a currently used polynomial id for the given level.*
  - carl::Variable `var` (std::size\_t level) const

*Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
  - bool `canBePurgedByBounds` (const `UPoly` &p) const

*Checks whether a polynomial can safely be ignored due to the bounds.*
  - bool `isPurged` (std::size\_t level, std::size\_t id)

## Protected Attributes

- `ProjectionInformation` `mInfo`

*Additional info on projection, projection levels and projection polynomials.*
- std::function< void(std::size\_t, const `SampleLiftedWith` &) > `mRemoveCallback`

*Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

## Friends

- template<typename S, Backtracking B>  
std::ostream & `operator<<` (std::ostream &os, const `Projection< Incrementality::FULL, B, S >` &p)

### 0.15.396.1 Constructor & Destructor Documentation

```
0.15.396.1.1 Projection() template<typename Settings , Backtracking BT>
smrat::cad::Projection< Incrementality::FULL, BT, Settings >::Projection (
 const Constraints & c) [inline]
```

### 0.15.396.2 Member Function Documentation

```
0.15.396.2.1 addEqConstraint() [1/2] template<typename Settings >
carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.396.2.2 addEqConstraint() [2/2] template<typename Settings >
virtual carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [inline], [virtual], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in `smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >`.

```
0.15.396.2.3 addPolynomial() [1/2] template<typename Settings >
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.396.2.4 addPolynomial() [2/2] template<typename Settings , Backtracking BT>
carl::Bitset smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::addPolynomial (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [inline], [override], [virtual]
```

Adds the given polynomial to the projection.

Implements [smtrat::cad::BaseProjection< Settings >](#).

```
0.15.396.2.5 callRemoveCallback() template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
 std::size_t level,
 const SampleLiftedWith & slw) const [inline], [protected], [inherited]
```

```
0.15.396.2.6 canBePurgedByBounds() template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
 const UPoly & p) const [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

```
0.15.396.2.7 dim() [1/2] template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::dim () const [inline], [inherited]
```

Returns the dimension of the projection.

```
0.15.396.2.8 dim() [2/2] template<typename Settings , Backtracking BT>
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

```
0.15.396.2.9 empty() [1/2] template<typename Settings >
virtual bool smtrat::cad::BaseProjection< Settings >::empty () [inline], [virtual], [inherited]
```

```
0.15.396.2.10 empty() [2/2] template<typename Settings , Backtracking BT>
bool smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::empty (
 std::size_t level) const [inline], [override], [virtual]
```

Implements [smtrat::cad::BaseProjection< Settings >](#).

```
0.15.396.2.11 exportAsDot() template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::FULL, BT, Settings >::exportAsDot (
 std::ostream & out) const [inline], [override], [virtual]
```

Reimplemented from [smtrat::cad::BaseProjection< Settings >](#).

**0.15.396.2.12 freeID()** template<typename Settings >  
void smrat::cad::BaseProjection< Settings >::freeID (  
 std::size\_t level,  
 std::size\_t id ) [inline], [protected], [inherited]  
Frees a currently used polynomial id for the given level.

**0.15.396.2.13 getID()** template<typename Settings >  
std::size\_t smrat::cad::BaseProjection< Settings >::getID (  
 std::size\_t level ) [inline], [protected], [inherited]  
Returns a fresh polynomial id for the given level.

**0.15.396.2.14 getOrigin()** template<typename Settings , Backtracking BT>  
Origin smrat::cad::Projection< Incrementality::FULL, BT, Settings >::getOrigin (  
 std::size\_t level,  
 std::size\_t id ) const [inline], [override], [virtual]  
Reimplemented from [smrat::cad::BaseProjection< Settings >](#).

**0.15.396.2.15 getPolyForLifting()** template<typename Settings >  
OptionalID smrat::cad::BaseProjection< Settings >::getPolyForLifting (  
 std::size\_t level,  
 SampleLiftedWith & slw ) [inline], [inherited]  
Get a polynomial from this level suited for lifting.

**0.15.396.2.16 getPolynomialById()** template<typename Settings , Backtracking BT>  
const UPoly& smrat::cad::Projection< Incrementality::FULL, BT, Settings >::getPolynomialById  
(  
 std::size\_t level,  
 std::size\_t id ) const [inline], [override], [virtual]  
Retrieves a polynomial from its id.  
Implements [smrat::cad::BaseProjection< Settings >](#).

**0.15.396.2.17 hasPolynomialById()** template<typename Settings , Backtracking BT>  
bool smrat::cad::Projection< Incrementality::FULL, BT, Settings >::hasPolynomialById (  
 std::size\_t level,  
 std::size\_t id ) const [inline], [override], [virtual]  
Implements [smrat::cad::BaseProjection< Settings >](#).

**0.15.396.2.18 isPurged()** template<typename Settings >  
bool smrat::cad::BaseProjection< Settings >::isPurged (  
 std::size\_t level,  
 std::size\_t id ) [inline], [protected], [inherited]

**0.15.396.2.19 projectNewPolynomial()** template<typename Settings , Backtracking BT>  
carl::Bitset smrat::cad::Projection< Incrementality::FULL, BT, Settings >::projectNewPolynomial  
(  
 const ConstraintSelection & cs = carl::Bitset(true) ) [inline]

```
0.15.396.2.20 removePolynomial() [1/2] template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::removePolynomial (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.396.2.21 removePolynomial() [2/2] template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::FULL , BT, Settings >::removePolynomial (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [inline], [override], [virtual]
```

Removes the given polynomial from the projection.

Implements [smtrat::cad::BaseProjection< Settings >](#).

```
0.15.396.2.22 reset() template<typename Settings , Backtracking BT>
void smtrat::cad::Projection< Incrementality::FULL , BT, Settings >::reset () [inline]
```

```
0.15.396.2.23 setRemoveCallback() template<typename Settings >
template<typename F >
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (
 F && f) [inline], [inherited]
```

Sets a callback that is called whenever polynomials are removed.

```
0.15.396.2.24 size() [1/4] template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::size () const [inline], [inherited]
```

```
0.15.396.2.25 size() [2/4] template<typename Settings , Backtracking BT>
std::size_t smtrat::cad::BaseProjection< Settings >::size (
 void) [inline]
```

```
0.15.396.2.26 size() [3/4] template<typename Settings , Backtracking BT>
std::size_t smtrat::cad::Projection< Incrementality::FULL , BT, Settings >::size (
 std::size_t level) const [inline], [override], [virtual]
```

Implements [smtrat::cad::BaseProjection< Settings >](#).

```
0.15.396.2.27 size() [4/4] template<typename Settings , Backtracking BT>
virtual std::size_t smtrat::cad::BaseProjection< Settings >::size (
 void)
```

```
0.15.396.2.28 var() template<typename Settings >
carl::Variable smtrat::cad::BaseProjection< Settings >::var (
 std::size_t level) const [inline], [protected], [inherited]
```

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.396.2.29 vars()** template<typename Settings >  
const auto& smrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]  
Returns the variables used for projection.

### 0.15.396.3 Friends And Related Function Documentation

**0.15.396.3.1 operator<<** template<typename Settings , Backtracking BT>  
template<typename S , Backtracking B>  
std::ostream& operator<< (  
 std::ostream & os,  
 const Projection< Incrementality::FULL, B, S > & p ) [friend]

### 0.15.396.4 Field Documentation

**0.15.396.4.1 mInfo** template<typename Settings >  
ProjectionInformation smrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]  
Additional info on projection, projection levels and projection polynomials.

**0.15.396.4.2 mRemoveCallback** template<typename Settings >  
std::function<void(std::size\_t, const SampleLiftedWith&)> smrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]  
Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

## 0.15.397 smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings > Class Template Reference

This class implements a projection that supports no incrementality and expects backtracking to be in order.

```
#include <Projection_NO.h>
```

### Public Member Functions

- **Projection** (const Constraints &c)  
• void **reset** ()  
*Resets all datastructures, use the given variables from now on.*
- carl::Bitset **addPolynomial** (const UPoly &p, std::size\_t cid, bool) override  
*Adds the given polynomial to the projection with the given constraint id as origin.*
- void **removePolynomial** (const UPoly &p, std::size\_t cid, bool) override  
*Removed the given polynomial from the projection.*
- std::size\_t **size** (std::size\_t level) const override  
*Returns the number of polynomials in this level.*
- bool **empty** (std::size\_t level) const override  
*Returns whether the number of polynomials in this level is zero.*
- carl::Bitset **projectNewPolynomial** (const ConstraintSelection &=carl::Bitset(true))  
*Returns false, as the projection is not incremental.*
- bool **hasPolynomialById** (std::size\_t level, std::size\_t id) const override
- const UPoly & **getPolynomialById** (std::size\_t level, std::size\_t id) const override  
*Get the polynomial from this level with the given id.*
- std::size\_t **dim** () const  
*Returns the dimension of the projection.*

- virtual std::size\_t `size` (std::size\_t level) const=0
- std::size\_t `size` () const
- std::size\_t `dim` () const
 

*Returns the dimension of the projection.*
- const auto & `vars` () const
 

*Returns the variables used for projection.*
- template<typename F>  
void `setRemoveCallback` (F &&f)
 

*Sets a callback that is called whenever polynomials are removed.*
- carl::Bitset `addPolynomial` (const Poly &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- carl::Bitset `addEqConstraint` (const Poly &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual carl::Bitset `addEqConstraint` (const UPoly &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial of an equational constraint to the projection.*
- void `removePolynomial` (const Poly &p, std::size\_t cid, bool isBound)
 

*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- std::size\_t `size` () const
- virtual bool `empty` ()
  - `OptionalID getPolyForLifting` (std::size\_t level, SampleLiftedWith &slw)
 

*Get a polynomial from this level suited for lifting.*
- virtual void `exportAsDot` (std::ostream &) const
- virtual Origin `getOrigin` (std::size\_t level, std::size\_t id) const

### Protected Member Functions

- void `callRemoveCallback` (std::size\_t level, const SampleLiftedWith &slw) const
- std::size\_t `getID` (std::size\_t level)
 

*Returns a fresh polynomial id for the given level.*
- void `freeID` (std::size\_t level, std::size\_t id)
 

*Frees a currently used polynomial id for the given level.*
- carl::Variable `var` (std::size\_t level) const
 

*Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
- bool `canBePurgedByBounds` (const UPoly &p) const
 

*Checks whether a polynomial can safely be ignored due to the bounds.*
- bool `isPurged` (std::size\_t level, std::size\_t id)

### Protected Attributes

- const Constraints & `mConstraints`
- ProjectionInformation `mInfo`

*Additional info on projection, projection levels and projection polynomials.*
- std::function< void(std::size\_t, const SampleLiftedWith &) > `mRemoveCallback`

*Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

### Friends

- template<typename S>  
std::ostream & `operator<<` (std::ostream &os, const Projection< Incrementality::NONE, Backtracking::ORDERED, S > &p)

### 0.15.397.1 Detailed Description

```
template<typename Settings>
class smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >
```

This class implements a projection that supports no incrementality and expects backtracking to be in order.  
It is based on the following data structures:

- mPolynomialIDs: maps polynomials to a (per level) unique id
- mPolynomials: stores polynomials as a list (per level) with their origin

The origin of a polynomial in level zero is the id of the corresponding constraint. For all other levels, it is the id of some polynomial from level zero such that the polynomial must be removed if the origin is removed. For a single projection operation, the resulting origin is the largest of the participating polynomials. If a polynomial is derived from multiple projection operations, the origin is the earliest and thus smallest, at least for this non-incremental setting.

### 0.15.397.2 Constructor & Destructor Documentation

```
0.15.397.2.1 Projection() template<typename Settings >
smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::Projection (
 const Constraints & c) [inline]
```

### 0.15.397.3 Member Function Documentation

```
0.15.397.3.1 addEqConstraint() [1/2] template<typename Settings >
carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.397.3.2 addEqConstraint() [2/2] template<typename Settings >
virtual carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [inline], [virtual], [inherited]
```

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in [smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >](#).

```
0.15.397.3.3 addPolynomial() [1/2] template<typename Settings >
carl::Bitset smrat::cad::BaseProjection< Settings >::addPolynomial (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.397.3.4 addPolynomial() [2/2] template<typename Settings >
carl::Bitset smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings
>::addPolynomial (
 const UPoly & p,
 std::size_t cid,
 bool) [inline], [override], [virtual]
```

Adds the given polynomial to the projection with the given constraint id as origin.

Asserts that the main variable of the polynomial is the first variable.

Implements [smtrat::cad::BaseProjection< Settings >](#).

```
0.15.397.3.5 callRemoveCallback() template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback (
 std::size_t level,
 const SampleLiftedWith & slw) const [inline], [protected], [inherited]
```

```
0.15.397.3.6 canBePurgedByBounds() template<typename Settings >
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds (
 const UPoly & p) const [inline], [protected], [inherited]
```

Checks whether a polynomial can safely be ignored due to the bounds.

```
0.15.397.3.7 dim() [1/2] template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::dim () const [inline], [inherited]
```

Returns the dimension of the projection.

```
0.15.397.3.8 dim() [2/2] template<typename Settings >
std::size_t smtrat::cad::BaseProjection< Settings >::dim [inline]
```

Returns the dimension of the projection.

```
0.15.397.3.9 empty() [1/2] template<typename Settings >
virtual bool smtrat::cad::BaseProjection< Settings >::empty () [inline], [virtual], [inherited]
```

```
0.15.397.3.10 empty() [2/2] template<typename Settings >
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::empty (
 std::size_t level) const [inline], [override], [virtual]
```

Returns whether the number of polynomials in this level is zero.

Implements [smtrat::cad::BaseProjection< Settings >](#).

```
0.15.397.3.11 exportAsDot() template<typename Settings >
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot (
 std::ostream &) const [inline], [virtual], [inherited]
```

Reimplemented in [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#), and [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#).

```
0.15.397.3.12 freeID() template<typename Settings >
void smtrat::cad::BaseProjection< Settings >::freeID (
 std::size_t level,
 std::size_t id) [inline], [protected], [inherited]
```

Frees a currently used polynomial id for the given level.

**0.15.397.3.13 `getID()`** template<typename `Settings`>  
std::size\_t `smtrat::cad::BaseProjection< Settings >`::`getID` (  
 std::size\_t `level`) [inline], [protected], [inherited]  
Returns a fresh polynomial id for the given level.

**0.15.397.3.14 `getOrigin()`** template<typename `Settings`>  
virtual `Origin` `smtrat::cad::BaseProjection< Settings >`::`getOrigin` (  
 std::size\_t `level`,  
 std::size\_t `id`) const [inline], [virtual], [inherited]  
Reimplemented in `smtrat::cad::Projection< Incrementality::FULL, BT, Settings >`.

**0.15.397.3.15 `getPolyForLifting()`** template<typename `Settings`>  
`OptionalID` `smtrat::cad::BaseProjection< Settings >`::`getPolyForLifting` (  
 std::size\_t `level`,  
 `SampleLiftedWith` & `slw`) [inline], [inherited]  
Get a polynomial from this level suited for lifting.

**0.15.397.3.16 `getPolynomialById()`** template<typename `Settings`>  
const `UPoly&` `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`::`getPolynomialById` (  
 std::size\_t `level`,  
 std::size\_t `id`) const [inline], [override], [virtual]  
Get the polynomial from this level with the given id.  
Implements `smtrat::cad::BaseProjection< Settings >`.

**0.15.397.3.17 `hasPolynomialById()`** template<typename `Settings`>  
bool `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`::`hasPolynomialById` (  
 std::size\_t `level`,  
 std::size\_t `id`) const [inline], [override], [virtual]  
Implements `smtrat::cad::BaseProjection< Settings >`.

**0.15.397.3.18 `isPurged()`** template<typename `Settings`>  
bool `smtrat::cad::BaseProjection< Settings >`::`isPurged` (  
 std::size\_t `level`,  
 std::size\_t `id`) [inline], [protected], [inherited]

**0.15.397.3.19 `projectNewPolynomial()`** template<typename `Settings`>  
carl::Bitset `smtrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >`::`projectNewPolynomial` (  
 const `ConstraintSelection` & = `carl::Bitset(true)`) [inline]  
Returns false, as the projection is not incremental.

**0.15.397.3.20 `removePolynomial()` [1/2]** template<typename `Settings`>  
void `smtrat::cad::BaseProjection< Settings >`::`removePolynomial` (  
 const `Poly` & `p`,  
 std::size\_t `cid`,  
 bool `isBound`) [inline], [inherited]

Removes the given polynomial from the projection. Converts to a `UPoly` and calls the appropriate overload.

**0.15.397.3.21 removePolynomial() [2/2]** template<typename Settings >  
void smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::remove←  
Polynomial ( const UPoly & p, std::size\_t cid, bool ) [inline], [override], [virtual]

Removed the given polynomial from the projection.

Asserts that this polynomial was the one added last and has the given constraint id as origin. Calls the callback function for every level with a mask designating the polynomials removed from this level.

Implements [smrat::cad::BaseProjection< Settings >](#).

**0.15.397.3.22 reset()** template<typename Settings >  
void smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::reset ( ) [inline]

Resets all datastructures, use the given variables from now on.

**0.15.397.3.23 setRemoveCallback()** template<typename Settings >  
template<typename F >  
void smrat::cad::BaseProjection< Settings >::setRemoveCallback ( F && f ) [inline], [inherited]

Sets a callback that is called whenever polynomials are removed.

**0.15.397.3.24 size() [1/4]** template<typename Settings >  
std::size\_t smrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]

**0.15.397.3.25 size() [2/4]** template<typename Settings >  
std::size\_t smrat::cad::BaseProjection< Settings >::size ( void ) [inline]

**0.15.397.3.26 size() [3/4]** template<typename Settings >  
std::size\_t smrat::cad::Projection< Incrementality::NONE, Backtracking::ORDERED, Settings >::size ( std::size\_t level ) const [inline], [override], [virtual]

Returns the number of polynomials in this level.

Implements [smrat::cad::BaseProjection< Settings >](#).

**0.15.397.3.27 size() [4/4]** template<typename Settings >  
virtual std::size\_t smrat::cad::BaseProjection< Settings >::size ( void )

**0.15.397.3.28 var()** template<typename Settings >  
carl::Variable smrat::cad::BaseProjection< Settings >::var ( std::size\_t level ) const [inline], [protected], [inherited]

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.397.3.29 vars()** template<typename Settings >  
const auto& smrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]

Returns the variables used for projection.

#### 0.15.397.4 Friends And Related Function Documentation

```
0.15.397.4.1 operator<< template<typename Settings >
template<typename S >
std::ostream& operator<< (
 std::ostream & os,
 const Projection< Incrementality::NONE, Backtracking::ORDERED, S > & p) [friend]
```

#### 0.15.397.5 Field Documentation

```
0.15.397.5.1 mConstraints template<typename Settings >
const Constraints& smrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]
```

**0.15.397.5.2 mInfo** template<typename Settings >  
*ProjectionInformation* smrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]  
Additional info on projection, projection levels and projection polynomials.

**0.15.397.5.3 mRemoveCallback** template<typename Settings >  
std::function<void(std::size\_t, const SampleLiftedWith&)> smrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]  
Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

### 0.15.398 smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings > Class Template Reference

This class implements a projection that supports no incrementality and allows backtracking to be out of order.  
`#include <Projection_NU.h>`

#### Public Member Functions

- `Projection` (const `Constraints` &c)
- `void reset()`
- `carl::Bitset addPolynomial` (const `UPoly` &p, `std::size_t` cid, `bool`) override  
*Adds the given polynomial to the projection.*
- `void removePolynomial` (const `UPoly` &, `std::size_t` cid, `bool`) override  
*Removes the given polynomial from the projection.*
- `std::size_t size` (`std::size_t` level) const override
- `bool empty` (`std::size_t` level) const override
- `carl::Bitset projectNewPolynomial` (const `ConstraintSelection` &=carl::Bitset(true))
- `bool hasPolynomialById` (`std::size_t` level, `std::size_t` id) const override
- `const UPoly & getPolynomialById` (`std::size_t` level, `std::size_t` id) const override  
*Retrieves a polynomial from its id.*
- `std::size_t dim()` const  
*Returns the dimension of the projection.*
- `virtual std::size_t size` (`std::size_t` level) const=0
- `std::size_t size()` const
- `std::size_t dim()` const  
*Returns the dimension of the projection.*
- `const auto & vars()` const

- *Returns the variables used for projection.*
- template<typename F >  
void **setRemoveCallback** (F &&f)  
*Sets a callback that is called whenever polynomials are removed.*
- carl::Bitset **addPolynomial** (const Poly &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- carl::Bitset **addEqConstraint** (const Poly &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.*
- virtual carl::Bitset **addEqConstraint** (const UPoly &p, std::size\_t cid, bool isBound)  
*Adds the given polynomial of an equational constraint to the projection.*
- void **removePolynomial** (const Poly &p, std::size\_t cid, bool isBound)  
*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- std::size\_t **size** () const
- virtual bool **empty** ()
- **OptionalID getPolyForLifting** (std::size\_t level, SampleLiftedWith &slw)  
*Get a polynomial from this level suited for lifting.*
- virtual void **exportAsDot** (std::ostream &) const
- virtual **Origin getOrigin** (std::size\_t level, std::size\_t id) const

### Protected Member Functions

- void **callRemoveCallback** (std::size\_t level, const SampleLiftedWith &slw) const  
*Returns a fresh polynomial id for the given level.*
- std::size\_t **getID** (std::size\_t level)  
*Returns a currently used polynomial id for the given level.*
- void **freeID** (std::size\_t level, std::size\_t id)  
*Frees a currently used polynomial id for the given level.*
- carl::Variable **var** (std::size\_t level) const  
*Returns the variable that corresponds to the given level, that is the variable eliminated in this level.*
- bool **canBePurgedByBounds** (const UPoly &p) const  
*Checks whether a polynomial can safely be ignored due to the bounds.*
- bool **isPurged** (std::size\_t level, std::size\_t id)

### Protected Attributes

- const Constraints & **mConstraints**
- **ProjectionInformation mInfo**  
*Additional info on projection, projection levels and projection polynomials.*
- std::function< void(std::size\_t, const SampleLiftedWith &) > **mRemoveCallback**  
*Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.*

### Friends

- template<typename S >  
std::ostream & **operator<<** (std::ostream &os, const Projection< Incrementality::NONE, Backtracking::UNORDERED, S > &p)

### 0.15.398.1 Detailed Description

```
template<typename Settings>
class smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >
```

This class implements a projection that supports no incrementality and allows backtracking to be out of order.

## 0.15.398.2 Constructor & Destructor Documentation

**0.15.398.2.1 Projection()** template<typename Settings >  
smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::Projection  
(  
    const Constraints & c ) [inline]

## 0.15.398.3 Member Function Documentation

**0.15.398.3.1 addEqConstraint() [1/2]** template<typename Settings >  
carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (  
    const Poly & p,  
    std::size\_t cid,  
    bool isBound ) [inline], [inherited]

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.398.3.2 addEqConstraint() [2/2]** template<typename Settings >  
virtual carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (  
    const UPoly & p,  
    std::size\_t cid,  
    bool isBound ) [inline], [virtual], [inherited]

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in smrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

**0.15.398.3.3 addPolynomial() [1/2]** template<typename Settings >  
carl::Bitset smrat::cad::BaseProjection< Settings >::addPolynomial (  
    const Poly & p,  
    std::size\_t cid,  
    bool isBound ) [inline], [inherited]

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.398.3.4 addPolynomial() [2/2]** template<typename Settings >  
carl::Bitset smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::addPolynomial (  
    const UPoly & p,  
    std::size\_t cid,  
    bool isBound ) [inline], [override], [virtual]

Adds the given polynomial to the projection.

Implements smrat::cad::BaseProjection< Settings >.

Reimplemented in smrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >.

**0.15.398.3.5 callRemoveCallback()** template<typename Settings >  
void smrat::cad::BaseProjection< Settings >::callRemoveCallback (  
    std::size\_t level,  
    const SampleLiftedWith & slw ) const [inline], [protected], [inherited]

**0.15.398.3.6 canBePurgedByBounds()** template<typename Settings >  
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds ( const UPoly & p ) const [inline], [protected], [inherited]  
Checks whether a polynomial can safely be ignored due to the bounds.

**0.15.398.3.7 dim() [1/2]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::dim () const [inline], [inherited]  
Returns the dimension of the projection.

**0.15.398.3.8 dim() [2/2]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::dim [inline]  
Returns the dimension of the projection.

**0.15.398.3.9 empty() [1/2]** template<typename Settings >  
virtual bool smtrat::cad::BaseProjection< Settings >::empty () [inline], [virtual], [inherited]

**0.15.398.3.10 empty() [2/2]** template<typename Settings >  
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::empty()  
( std::size\_t level ) const [inline], [override], [virtual]  
Implements smtrat::cad::BaseProjection< Settings >.

**0.15.398.3.11 exportAsDot()** template<typename Settings >  
virtual void smtrat::cad::BaseProjection< Settings >::exportAsDot ( std::ostream & ) const [inline], [virtual], [inherited]  
Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >, and smtrat::cad::Projection< Incrementality::FULL,

**0.15.398.3.12 freeID()** template<typename Settings >  
void smtrat::cad::BaseProjection< Settings >::freeID ( std::size\_t level, std::size\_t id ) [inline], [protected], [inherited]  
Frees a currently used polynomial id for the given level.

**0.15.398.3.13 getID()** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::getID ( std::size\_t level ) [inline], [protected], [inherited]  
Returns a fresh polynomial id for the given level.

**0.15.398.3.14 getOrigin()** template<typename Settings >  
virtual Origin smtrat::cad::BaseProjection< Settings >::getOrigin ( std::size\_t level, std::size\_t id ) const [inline], [virtual], [inherited]  
Reimplemented in smtrat::cad::Projection< Incrementality::FULL, BT, Settings >.

```
0.15.398.3.15 getPolyForLifting() template<typename Settings >
OptionalID smrat::cad::BaseProjection< Settings >::getPolyForLifting (
 std::size_t level,
 SampleLiftedWith & slw) [inline], [inherited]
```

Get a polynomial from this level suited for lifting.

```
0.15.398.3.16 getPolynomialById() template<typename Settings >
const UPoly& smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings
>::getPolynomialById (
 std::size_t level,
 std::size_t id) const [inline], [override], [virtual]
```

Retrieves a polynomial from its id.

Implements [smrat::cad::BaseProjection< Settings >](#).

```
0.15.398.3.17 hasPolynomialById() template<typename Settings >
bool smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::has←
PolynomialById (
 std::size_t level,
 std::size_t id) const [inline], [override], [virtual]
```

Implements [smrat::cad::BaseProjection< Settings >](#).

```
0.15.398.3.18 isPurged() template<typename Settings >
bool smrat::cad::BaseProjection< Settings >::isPurged (
 std::size_t level,
 std::size_t id) [inline], [protected], [inherited]
```

```
0.15.398.3.19 projectNewPolynomial() template<typename Settings >
carl::Bitset smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings
>::projectNewPolynomial (
 const ConstraintSelection & = carl::Bitset(true)) [inline]
```

```
0.15.398.3.20 removePolynomial() [1/2] template<typename Settings >
void smrat::cad::BaseProjection< Settings >::removePolynomial (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

```
0.15.398.3.21 removePolynomial() [2/2] template<typename Settings >
void smrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >↔
::removePolynomial (
 const UPoly & p,
 std::size_t cid,
 bool isBound) [inline], [override], [virtual]
```

Removes the given polynomial from the projection.

Implements [smrat::cad::BaseProjection< Settings >](#).

Reimplemented in [smrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >](#).

**0.15.398.3.22 `reset()`** template<typename Settings >  
void `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`::reset  
( ) [inline]

**0.15.398.3.23 `setRemoveCallback()`** template<typename Settings >  
template<typename F >  
void `smtrat::cad::BaseProjection< Settings >`::setRemoveCallback ( F && f ) [inline], [inherited]  
Sets a callback that is called whenever polynomials are removed.

**0.15.398.3.24 `size() [1/4]`** template<typename Settings >  
std::size\_t `smtrat::cad::BaseProjection< Settings >`::size ( ) const [inline], [inherited]

**0.15.398.3.25 `size() [2/4]`** template<typename Settings >  
std::size\_t `smtrat::cad::BaseProjection< Settings >`::size ( void ) [inline]

**0.15.398.3.26 `size() [3/4]`** template<typename Settings >  
std::size\_t `smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >`::size ( std::size\_t level ) const [inline], [override], [virtual]  
Implements `smtrat::cad::BaseProjection< Settings >`.

**0.15.398.3.27 `size() [4/4]`** template<typename Settings >  
virtual std::size\_t `smtrat::cad::BaseProjection< Settings >`::size ( void )

**0.15.398.3.28 `var()`** template<typename Settings >  
carl::Variable `smtrat::cad::BaseProjection< Settings >`::var ( std::size\_t level ) const [inline], [protected], [inherited]  
Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.398.3.29 `vars()`** template<typename Settings >  
const auto& `smtrat::cad::BaseProjection< Settings >`::vars ( ) const [inline], [inherited]  
Returns the variables used for projection.

## 0.15.398.4 Friends And Related Function Documentation

**0.15.398.4.1 `operator<<`** template<typename Settings >  
template<typename S >  
std::ostream& operator<< ( std::ostream & os, const `Projection< Incrementality::NONE, Backtracking::UNORDERED, S >`& p ) [friend]

## 0.15.398.5 Field Documentation

---

**0.15.398.5.1 mConstraints** template<typename Settings >  
const `Constraints&` `smtrat::cad::BaseProjection< Settings >::mConstraints` [protected], [inherited]

**0.15.398.5.2 mInfo** template<typename Settings >  
`ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo` [protected], [inherited]  
Additional info on projection, projection levels and projection polynomials.

**0.15.398.5.3 mRemoveCallback** template<typename Settings >  
std::function<void(std::size\_t, const `SampleLiftedWith`&)> `smtrat::cad::BaseProjection< Settings >::mRemoveCallback` [protected], [inherited]  
Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

## 0.15.399 `smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >` Class Template Reference

```
#include <Projection_S.h>
```

### Public Member Functions

- template<typename ... Args> **Projection** (Args &&... args)
- void **reset** ()
- carl::Bitset **addPolynomial** (const `UPoly` &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial to the projection.*
- void **removePolynomial** (const `UPoly` &p, std::size\_t cid, bool isBound)
 

*Removes the given polynomial from the projection.*
- carl::Bitset **projectNewPolynomial** (const `ConstraintSelection` &=carl::Bitset(true))
- carl::Bitset **addPolynomial** (const `Poly` &p, std::size\_t cid, bool isBound)
 

*Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.*
- void **removePolynomial** (const `Poly` &p, std::size\_t cid, bool isBound)
 

*Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.*
- std::size\_t **size** (std::size\_t level) const override
- virtual std::size\_t **size** (std::size\_t level) const=0
- std::size\_t **size** () const
- std::size\_t **size** () const
- bool **empty** (std::size\_t level) const override
- virtual bool **empty** ()
- bool **hasPolynomialById** (std::size\_t level, std::size\_t id) const override
- const `UPoly` & **getPolynomialById** (std::size\_t level, std::size\_t id) const override
 

*Retrieves a polynomial from its id.*
- std::size\_t **dim** () const
 

*Returns the dimension of the projection.*
- std::size\_t **dim** () const
 

*Returns the dimension of the projection.*
- const auto & **vars** () const
 

*Returns the variables used for projection.*
- template<typename F >
 void **setRemoveCallback** (F &&f)
 

*Sets a callback that is called whenever polynomials are removed.*
- carl::Bitset **addEqConstraint** (const `Poly` &p, std::size\_t cid, bool isBound)

- virtual carl::Bitset **addEqConstraint** (const UPoly &p, std::size\_t cid, bool isBound)
 

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.
- virtual OptionalID **getPolyForLifting** (std::size\_t level, SampleLiftedWith &slw)
 

Get a polynomial from this level suited for lifting.
- virtual void **exportAsDot** (std::ostream &) const
- virtual Origin **getOrigin** (std::size\_t level, std::size\_t id) const

### Protected Member Functions

- void **callRemoveCallback** (std::size\_t level, const SampleLiftedWith &slw) const
- std::size\_t **getID** (std::size\_t level)
 

Returns a fresh polynomial id for the given level.
- void **freeID** (std::size\_t level, std::size\_t id)
 

Frees a currently used polynomial id for the given level.
- carl::Variable **var** (std::size\_t level) const
 

Returns the variable that corresponds to the given level, that is the variable eliminated in this level.
- bool **canBePurgedByBounds** (const UPoly &p) const
 

Checks whether a polynomial can safely be ignored due to the bounds.
- bool **isPurged** (std::size\_t level, std::size\_t id)

### Protected Attributes

- const Constraints & **mConstraints**
- ProjectionInformation **mInfo**

Additional info on projection, projection levels and projection polynomials.
- std::function< void(std::size\_t, const SampleLiftedWith &) > **mRemoveCallback**

Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

### Friends

- template<typename S , Backtracking B>
 std::ostream & **operator<<** (std::ostream &os, const Projection< Incrementality::SIMPLE, B, S > &p)

## 0.15.399.1 Constructor & Destructor Documentation

```
0.15.399.1.1 Projection() template<typename Settings , Backtracking BT>
template<typename ... Args>
smrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::Projection (
 Args &&... args) [inline]
```

## 0.15.399.2 Member Function Documentation

```
0.15.399.2.1 addEqConstraint() [1/2] template<typename Settings >
carl::Bitset smrat::cad::BaseProjection< Settings >::addEqConstraint (
 const Poly & p,
 std::size_t cid,
 bool isBound) [inline], [inherited]
```

Adds the given polynomial of an equational constraint to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.399.2.2 addEqConstraint() [2/2]** template<typename Settings >  
virtual carl::Bitset smtrat::cad::BaseProjection< Settings >::addEqConstraint ( const UPoly & p, std::size\_t cid, bool isBound ) [inline], [virtual], [inherited]

Adds the given polynomial of an equational constraint to the projection.

Reimplemented in smtrat::cad::Projection< Incrementality::FULL, Backtracking::HIDE, Settings >.

**0.15.399.2.3 addPolynomial() [1/2]** template<typename Settings >  
carl::Bitset smtrat::cad::BaseProjection< Settings >::addPolynomial ( const Poly & p, std::size\_t cid, bool isBound ) [inline], [inherited]

Adds the given polynomial to the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.399.2.4 addPolynomial() [2/2]** template<typename Settings , Backtracking BT>  
carl::Bitset smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::addPolynomial ( const UPoly & p, std::size\_t cid, bool isBound ) [inline], [virtual]

Adds the given polynomial to the projection.

Reimplemented from smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >.

**0.15.399.2.5 callRemoveCallback()** template<typename Settings >  
void smtrat::cad::BaseProjection< Settings >::callRemoveCallback ( std::size\_t level, const SampleLiftedWith & slw ) const [inline], [protected], [inherited]

**0.15.399.2.6 canBePurgedByBounds()** template<typename Settings >  
bool smtrat::cad::BaseProjection< Settings >::canBePurgedByBounds ( const UPoly & p ) const [inline], [protected], [inherited]

Checks whether a polynomial can safely be ignored due to the bounds.

**0.15.399.2.7 dim() [1/2]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::dim ( ) const [inline], [inherited]  
Returns the dimension of the projection.

**0.15.399.2.8 dim() [2/2]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::dim [inline], [inherited]  
Returns the dimension of the projection.

**0.15.399.2.9 empty() [1/2]** template<typename Settings >  
virtual bool smtrat::cad::BaseProjection< Settings >::empty ( ) [inline], [virtual], [inherited]

**0.15.399.2.10 empty() [2/2]** template<typename Settings >  
bool smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::empty ( std::size\_t level ) const [inline], [override], [virtual], [inherited]

Implements [smtrat::cad::BaseProjection< Settings >](#).

**0.15.399.2.11 `exportAsDot()`** template<typename Settings >  
 virtual void [smtrat::cad::BaseProjection< Settings >](#)::exportAsDot ( std::ostream & ) const [inline], [virtual], [inherited]

Reimplemented in [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#), and [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#).

**0.15.399.2.12 `freeID()`** template<typename Settings >  
 void [smtrat::cad::BaseProjection< Settings >](#)::freeID ( std::size\_t level, std::size\_t id ) [inline], [protected], [inherited]

Frees a currently used polynomial id for the given level.

**0.15.399.2.13 `getID()`** template<typename Settings >  
 std::size\_t [smtrat::cad::BaseProjection< Settings >](#)::getID ( std::size\_t level ) [inline], [protected], [inherited]

Returns a fresh polynomial id for the given level.

**0.15.399.2.14 `getOrigin()`** template<typename Settings >  
 virtual Origin [smtrat::cad::BaseProjection< Settings >](#)::getOrigin ( std::size\_t level, std::size\_t id ) const [inline], [virtual], [inherited]

Reimplemented in [smtrat::cad::Projection< Incrementality::FULL, BT, Settings >](#).

**0.15.399.2.15 `getPolyForLifting()`** template<typename Settings >  
 OptionalID [smtrat::cad::BaseProjection< Settings >](#)::getPolyForLifting ( std::size\_t level, SampleLiftedWith & slw ) [inline], [inherited]

Get a polynomial from this level suited for lifting.

**0.15.399.2.16 `getPolynomialById()`** template<typename Settings >  
 const UPoly& [smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >](#)::getPolynomialById ( std::size\_t level, std::size\_t id ) const [inline], [override], [virtual], [inherited]

Retrieves a polynomial from its id.

Implements [smtrat::cad::BaseProjection< Settings >](#).

**0.15.399.2.17 `hasPolynomialById()`** template<typename Settings >  
 bool [smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >](#)::hasPolynomialById ( std::size\_t level, std::size\_t id ) const [inline], [override], [virtual], [inherited]

Implements [smtrat::cad::BaseProjection< Settings >](#).

**0.15.399.2.18 `isPurged()`** template<typename Settings >  
 bool [smtrat::cad::BaseProjection< Settings >](#)::isPurged (

```
 std::size_t level,
 std::size_t id) [inline], [protected], [inherited]
```

**0.15.399.2.19 `projectNewPolynomial()`** template<typename Settings , Backtracking BT>  
carl::Bitset smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::projectNew←  
Polynomial (   
 const ConstraintSelection & = carl::Bitset(true) ) [inline]

**0.15.399.2.20 `removePolynomial()` [1/2]** template<typename Settings >  
void smtrat::cad::BaseProjection< Settings >::removePolynomial (   
 const Poly & p,  
 std::size\_t cid,  
 bool isBound ) [inline], [inherited]

Removes the given polynomial from the projection. Converts to a UPoly and calls the appropriate overload.

**0.15.399.2.21 `removePolynomial()` [2/2]** template<typename Settings , Backtracking BT>  
void smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::removePolynomial (   
 const UPoly & p,  
 std::size\_t cid,  
 bool isBound ) [inline], [virtual]

Removes the given polynomial from the projection.

Reimplemented from [smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >](#).

**0.15.399.2.22 `reset()`** template<typename Settings , Backtracking BT>  
void smtrat::cad::Projection< Incrementality::SIMPLE, BT, Settings >::reset ( ) [inline]

**0.15.399.2.23 `setRemoveCallback()`** template<typename Settings >  
template<typename F >  
void smtrat::cad::BaseProjection< Settings >::setRemoveCallback (   
 F && f ) [inline], [inherited]

Sets a callback that is called whenever polynomials are removed.

**0.15.399.2.24 `size()` [1/4]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::size ( ) const [inline], [inherited]

**0.15.399.2.25 `size()` [2/4]** template<typename Settings >  
std::size\_t smtrat::cad::BaseProjection< Settings >::size (   
 void ) [inline], [inherited]

**0.15.399.2.26 `size()` [3/4]** template<typename Settings >  
std::size\_t smtrat::cad::Projection< Incrementality::NONE, Backtracking::UNORDERED, Settings >::size (   
 std::size\_t level ) const [inline], [override], [virtual], [inherited]  
Implements [smtrat::cad::BaseProjection< Settings >](#).

**0.15.399.2.27 `size()` [4/4]** template<typename Settings >  
 virtual std::size\_t smtrat::cad::BaseProjection< Settings >::size ( void ) [inherited]

**0.15.399.2.28 `var()`** template<typename Settings >  
 carl::Variable smtrat::cad::BaseProjection< Settings >::var ( std::size\_t level ) const [inline], [protected], [inherited]  
 Returns the variable that corresponds to the given level, that is the variable eliminated in this level.

**0.15.399.2.29 `vars()`** template<typename Settings >  
 const auto& smtrat::cad::BaseProjection< Settings >::vars ( ) const [inline], [inherited]  
 Returns the variables used for projection.

### 0.15.399.3 Friends And Related Function Documentation

**0.15.399.3.1 `operator<<`** template<typename Settings , Backtracking BT>  
 template<typename S , Backtracking B>  
 std::ostream& operator<< ( std::ostream & os, const Projection< Incrementality::SIMPLE, B, S > & p ) [friend]

### 0.15.399.4 Field Documentation

**0.15.399.4.1 `mConstraints`** template<typename Settings >  
 const Constraints& smtrat::cad::BaseProjection< Settings >::mConstraints [protected], [inherited]

**0.15.399.4.2 `mInfo`** template<typename Settings >  
 ProjectionInformation smtrat::cad::BaseProjection< Settings >::mInfo [protected], [inherited]  
 Additional info on projection, projection levels and projection polynomials.

**0.15.399.4.3 `mRemoveCallback`** template<typename Settings >  
 std::function<void(std::size\_t, const SampleLiftedWith&)> smtrat::cad::BaseProjection< Settings >::mRemoveCallback [protected], [inherited]  
 Callback to be called when polynomials are removed. The arguments are the projection level and a bitset that indicate which polynomials were removed in this level.

## 0.15.400 smtrat::cad::projection\_compare::ProjectionComparator< Strategy > Struct Template Reference

```
#include <ProjectionComparator.h>
```

### 0.15.401 smtrat::cad::projection\_compare::ProjectionComparator< ProjectionCompareStrategy::D > Struct Reference

```
#include <ProjectionComparator.h>
```

#### Public Member Functions

- bool `operator()` (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

### 0.15.401.1 Member Function Documentation

```
0.15.401.1.1 operator() bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args
>::operator() (
 const Candidate< Poly > & lhs,
 const Candidate< Poly > & rhs) const [inline], [inherited]
```

## 0.15.402 smtrat::cad::projection\_compare::ProjectionComparator< ProjectionCompareStrategy::LD > Struct Reference

```
#include <ProjectionComparator.h>
```

### Public Member Functions

- bool `operator()` (const `Candidate< Poly >` &lhs, const `Candidate< Poly >` &rhs) const

### 0.15.402.1 Member Function Documentation

```
0.15.402.1.1 operator() bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args
>::operator() (
 const Candidate< Poly > & lhs,
 const Candidate< Poly > & rhs) const [inline], [inherited]
```

## 0.15.403 smtrat::cad::projection\_compare::ProjectionComparator< ProjectionCompareStrategy::ID > Struct Reference

```
#include <ProjectionComparator.h>
```

### Public Member Functions

- bool `operator()` (const `Candidate< Poly >` &lhs, const `Candidate< Poly >` &rhs) const

### 0.15.403.1 Member Function Documentation

```
0.15.403.1.1 operator() bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args
>::operator() (
 const Candidate< Poly > & lhs,
 const Candidate< Poly > & rhs) const [inline], [inherited]
```

## 0.15.404 smtrat::cad::projection\_compare::ProjectionComparator< ProjectionCompareStrategy::PD > Struct Reference

```
#include <ProjectionComparator.h>
```

### Public Member Functions

- bool `operator()` (const `Candidate< Poly >` &lhs, const `Candidate< Poly >` &rhs) const

### 0.15.404.1 Member Function Documentation

```
0.15.404.1.1 operator() bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
 const Candidate< Poly > & lhs,
 const Candidate< Poly > & rhs) const [inline], [inherited]
```

## 0.15.405 smtrat::cad::projection\_compare::ProjectionComparator< ProjectionCompareStrategy::SD > Struct Reference

```
#include <ProjectionComparator.h>
```

### Public Member Functions

- bool **operator()** (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

#### 0.15.405.1 Member Function Documentation

```
0.15.405.1.1 operator() bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
 const Candidate< Poly > & lhs,
 const Candidate< Poly > & rhs) const [inline], [inherited]
```

## 0.15.406 smtrat::cad::projection\_compare::ProjectionComparator\_impl< Args > Struct Template Reference

```
#include <ProjectionComparator.h>
```

### Public Member Functions

- template<typename Poly >
 bool **operator()** (const Candidate< Poly > &lhs, const Candidate< Poly > &rhs) const

#### 0.15.406.1 Member Function Documentation

```
0.15.406.1.1 operator() template<typename... Args>
template<typename Poly >
bool smtrat::cad::projection_compare::ProjectionComparator_impl< Args >::operator() (
 const Candidate< Poly > & lhs,
 const Candidate< Poly > & rhs) const [inline]
```

## 0.15.407 smtrat::cad::ProjectionGlobalInformation Class Reference

```
#include <ProjectionInformation.h>
```

### Data Structures

- struct **ECData**

### Public Types

- using **ECMap** = std::map< Origin::BaseType, ECData >

## Public Member Functions

- void `reset` (std::size\_t dim)
- void `createEC` (const `Origin::BaseType` &origin)
- bool `isEC` (const `Origin::BaseType` &origin) const
- void `addToEC` (const `Origin::BaseType` &origin, std::size\_t level, std::size\_t pid)
- void `removeEC` (const `Origin::BaseType` &origin)

## Data Fields

- `ECMap mECs`
- `std::vector< ECMap::const_iterator > mUsedEC`
- `carl::Bitset mInactive`

### 0.15.407.1 Member Typedef Documentation

**0.15.407.1.1 `ECMap`** using `smtrat::cad::ProjectionGlobalInformation::ECMap = std::map<Origin::BaseType, ECData>`

### 0.15.407.2 Member Function Documentation

**0.15.407.2.1 `addToEC()`** void `smtrat::cad::ProjectionGlobalInformation::addToEC` (const `Origin::BaseType` & `origin`, std::size\_t `level`, std::size\_t `pid`) [inline]

**0.15.407.2.2 `createEC()`** void `smtrat::cad::ProjectionGlobalInformation::createEC` (const `Origin::BaseType` & `origin`) [inline]

**0.15.407.2.3 `isEC()`** bool `smtrat::cad::ProjectionGlobalInformation::isEC` (const `Origin::BaseType` & `origin`) const [inline]

**0.15.407.2.4 `removeEC()`** void `smtrat::cad::ProjectionGlobalInformation::removeEC` (const `Origin::BaseType` & `origin`) [inline]

**0.15.407.2.5 `reset()`** void `smtrat::cad::ProjectionGlobalInformation::reset` (std::size\_t `dim`) [inline]

### 0.15.407.3 Field Documentation

**0.15.407.3.1 `mECs`** `ECMap smtrat::cad::ProjectionGlobalInformation::mECs`

**0.15.407.3.2 `mInactive`** `carl::Bitset smtrat::cad::ProjectionGlobalInformation::mInactive`

**0.15.407.3.3 mUsedEC** std::vector<ECMap::const\_iterator> smtrat::cad::ProjectionGlobal<Information>::mUsedEC

## 0.15.408 smtrat::cad::ProjectionInformation Class Reference

```
#include <ProjectionInformation.h>
```

### Public Member Functions

- const auto & **operator()** () const
- auto & **operator()** ()
- const auto & **operator()** (std::size\_t level) const
- auto & **operator()** (std::size\_t level)
- bool **hasInfo** (std::size\_t level) const
- const auto & **operator()** (std::size\_t level, std::size\_t pid) const
- auto & **operator()** (std::size\_t level, std::size\_t pid)
- void **clear** (std::size\_t level, std::size\_t pid)
- void **emplace** (std::size\_t level, std::size\_t pid)
- bool **hasInfo** (std::size\_t level, std::size\_t pid) const
- bool **isActive** (std::size\_t level, std::size\_t id) const
- void **reset** (std::size\_t dim)
- void **addECConstraint** (std::size\_t pid)
- void **removeECConstraint** (std::size\_t)
- bool **hasEC** (std::size\_t level) const
- bool **usingEC** (std::size\_t level) const
- const carl::Bitset & **getUsedEC** (std::size\_t level) const
- bool **selectEC** (std::size\_t level)
- void **unselectEC** (std::size\_t level)

### 0.15.408.1 Member Function Documentation

**0.15.408.1.1 addECConstraint()** void smtrat::cad::ProjectionInformation::addECConstraint ( std::size\_t pid ) [inline]

**0.15.408.1.2 clear()** void smtrat::cad::ProjectionInformation::clear ( std::size\_t level, std::size\_t pid ) [inline]

**0.15.408.1.3 emplace()** void smtrat::cad::ProjectionInformation::emplace ( std::size\_t level, std::size\_t pid ) [inline]

**0.15.408.1.4 getUsedEC()** const carl::Bitset& smtrat::cad::ProjectionInformation::getUsedEC ( std::size\_t level ) const [inline]

**0.15.408.1.5 hasEC()** bool smtrat::cad::ProjectionInformation::hasEC ( std::size\_t level ) const [inline]

**0.15.408.1.6 `hasInfo()` [1/2]** `bool smtrat::cad::ProjectionInformation::hasInfo ( std::size_t level ) const [inline]`

**0.15.408.1.7 `hasInfo()` [2/2]** `bool smtrat::cad::ProjectionInformation::hasInfo ( std::size_t level, std::size_t pid ) const [inline]`

**0.15.408.1.8 `isActive()`** `bool smtrat::cad::ProjectionInformation::isActive ( std::size_t level, std::size_t id ) const [inline]`

**0.15.408.1.9 `operator()()` [1/6]** `auto& smtrat::cad::ProjectionInformation::operator() ( ) [inline]`

**0.15.408.1.10 `operator()()` [2/6]** `const auto& smtrat::cad::ProjectionInformation::operator() ( ) const [inline]`

**0.15.408.1.11 `operator()()` [3/6]** `auto& smtrat::cad::ProjectionInformation::operator() ( std::size_t level ) [inline]`

**0.15.408.1.12 `operator()()` [4/6]** `const auto& smtrat::cad::ProjectionInformation::operator() ( std::size_t level ) const [inline]`

**0.15.408.1.13 `operator()()` [5/6]** `auto& smtrat::cad::ProjectionInformation::operator() ( std::size_t level, std::size_t pid ) [inline]`

**0.15.408.1.14 `operator()()` [6/6]** `const auto& smtrat::cad::ProjectionInformation::operator() ( std::size_t level, std::size_t pid ) const [inline]`

**0.15.408.1.15 `removeECConstraint()`** `void smtrat::cad::ProjectionInformation::removeECConstraint ( std::size_t ) [inline]`

**0.15.408.1.16 `reset()`** `void smtrat::cad::ProjectionInformation::reset ( std::size_t dim ) [inline]`

**0.15.408.1.17 `selectEC()`** `bool smtrat::cad::ProjectionInformation::selectEC ( std::size_t level ) [inline]`

**0.15.408.1.18 `unselectEC()`** `void smtrat::cad::ProjectionInformation::unselectEC ( std::size_t level ) [inline]`

```
0.15.408.1.19 usingEC() bool smtrat::cad::ProjectionInformation::usingEC (
 std::size_t level) const [inline]
```

## 0.15.409 smtrat::cad::ProjectionLevelInformation Class Reference

```
#include <ProjectionInformation.h>
```

### Data Structures

- struct [LevelInfo](#)

### Public Member Functions

- bool [hasInfo](#) (std::size\_t level) const
- void [emplace](#) (std::size\_t level)
- const auto & [operator\(\)](#) (std::size\_t level) const
- auto & [operator\(\)](#) (std::size\_t level)
- void [reset](#) (std::size\_t dim)

#### 0.15.409.1 Member Function Documentation

```
0.15.409.1.1 emplace() void smtrat::cad::ProjectionLevelInformation::emplace (
 std::size_t level) [inline]
```

```
0.15.409.1.2 hasInfo() bool smtrat::cad::ProjectionLevelInformation::hasInfo (
 std::size_t level) const [inline]
```

```
0.15.409.1.3 operator()() [1/2] auto& smtrat::cad::ProjectionLevelInformation::operator() (
 std::size_t level) [inline]
```

```
0.15.409.1.4 operator()() [2/2] const auto& smtrat::cad::ProjectionLevelInformation::operator() (
 std::size_t level) const [inline]
```

```
0.15.409.1.5 reset() void smtrat::cad::ProjectionLevelInformation::reset (
 std::size_t dim) [inline]
```

## 0.15.410 smtrat::cad::ProjectionOperator Struct Reference

```
#include <ProjectionOperator.h>
```

### Public Member Functions

- template<typename Callback>  
void [operator\(\)](#) (ProjectionType pt, const UPoly &p, carl::Variable variable, Callback &&cb) const
- template<typename Callback>  
void [operator\(\)](#) (ProjectionType pt, const UPoly &p, const UPoly &q, carl::Variable variable, Callback &&i) const

#### 0.15.410.1 Member Function Documentation

```
0.15.410.1.1 operator() [1/2] template<typename Callback >
void smrat::cad::ProjectionOperator::operator() (
 ProjectionType pt,
 const UPoly & p,
 carl::Variable variable,
 Callback && cb) const [inline]
```

```
0.15.410.1.2 operator() [2/2] template<typename Callback >
void smrat::cad::ProjectionOperator::operator() (
 ProjectionType pt,
 const UPoly & p,
 const UPoly & q,
 carl::Variable variable,
 Callback && i) const [inline]
```

## 0.15.411 smrat::cad::ProjectionPolynomialInformation Class Reference

```
#include <ProjectionInformation.h>
```

### Data Structures

- struct [PolyInfo](#)

### Public Member Functions

- bool [hasInfo](#) (std::size\_t level, std::size\_t pid) const
- void [emplace](#) (std::size\_t level, std::size\_t pid)
- const auto & [operator\(\)](#) (std::size\_t level, std::size\_t pid) const
- auto & [operator\(\)](#) (std::size\_t level, std::size\_t pid)
- void [clear](#) (std::size\_t level, std::size\_t pid)
- void [reset](#) (std::size\_t)

### 0.15.411.1 Member Function Documentation

```
0.15.411.1.1 clear() void smrat::cad::ProjectionPolynomialInformation::clear (
 std::size_t level,
 std::size_t pid) [inline]
```

```
0.15.411.1.2 emplace() void smrat::cad::ProjectionPolynomialInformation::emplace (
 std::size_t level,
 std::size_t pid) [inline]
```

```
0.15.411.1.3 hasInfo() bool smrat::cad::ProjectionPolynomialInformation::hasInfo (
 std::size_t level,
 std::size_t pid) const [inline]
```

```
0.15.411.1.4 operator() [1/2] auto& smrat::cad::ProjectionPolynomialInformation::operator() (
 std::size_t level,
 std::size_t pid) [inline]
```

```
0.15.411.1.5 operator() [2/2] const auto& smtrat::cad::ProjectionPolynomialInformation::operator()
(
 std::size_t level,
 std::size_t pid) const [inline]
```

```
0.15.411.1.6 reset() void smtrat::cad::ProjectionPolynomialInformation::reset (
 std::size_t) [inline]
```

## 0.15.412 smtrat::cadcells::datastructures::Projections Class Reference

Encapsulates all computations on polynomials.

```
#include <projections.h>
```

### Public Member Functions

- auto [main\\_var](#) (PolyRef p) const
- [Projections](#) (PolyPool &pool)
- auto & [polys](#) ()
- const auto & [polys](#) () const
- void [clear\\_cache](#) (size\_t level)
 

*Clears all polynomials of the specified level and higher in the polynomial cache as well as their projection results.*
- void [clear\\_assignment\\_cache](#) (const Assignment &assignment)
 

*Clears all projections cached with respect to this assignment.*
- PolyRef [res](#) (PolyRef p, PolyRef q)
- bool [know\\_disc](#) (PolyRef p) const
- bool [known](#) (const Polynomial &p) const
- PolyRef [disc](#) (PolyRef p)
- PolyRef [ldcf](#) (PolyRef p)
- const std::vector< PolyRef > & [factors\\_nonconst](#) (PolyRef p)
- bool [is\\_zero](#) (const Assignment &sample, PolyRef p)
- size\_t [num\\_roots](#) (const Assignment &sample, PolyRef p)
- std::vector< RAN > [real\\_roots](#) (const Assignment &sample, PolyRef p)
- RAN [evaluate](#) (const Assignment &sample, IndexedRoot r)
- bool [is\\_nullified](#) (const Assignment &sample, PolyRef p)
- bool [is\\_ldcf\\_zero](#) (const Assignment &sample, PolyRef p)
- bool [is\\_disc\\_zero](#) (const Assignment &sample, PolyRef p)
- bool [is\\_const](#) (PolyRef p)
- bool [is\\_zero](#) (PolyRef p)
- bool [has\\_const\\_coeff](#) (PolyRef p) const
- PolyRef [simplest\\_nonzero\\_coeff](#) (const Assignment &sample, PolyRef p, std::function< bool(const Polynomial &, const Polynomial &) > compare) const
- std::size\_t [degree](#) (PolyRef p)

### 0.15.412.1 Detailed Description

Encapsulates all computations on polynomials.

Computations are cached with respect to a [PolyPool](#).

### 0.15.412.2 Constructor & Destructor Documentation

```
0.15.412.2.1 Projections() smtrat::cadcells::datastructures::Projections::Projections (
 PolyPool & pool) [inline]
```

### 0.15.412.3 Member Function Documentation

**0.15.412.3.1 `clear_assignment_cache()`** void smtrat::cadcells::datastructures::Projections::  
::clear\_assignment\_cache ( const Assignment & assignment ) [inline]

Clears all projections cached with respect to this assignment.

**0.15.412.3.2 `clear_cache()`** void smtrat::cadcells::datastructures::Projections::clear\_cache ( size\_t level ) [inline]

Clears all polynomials of the specified level and higher in the polynomial cache as well as their projection results.

**0.15.412.3.3 `degree()`** std::size\_t smtrat::cadcells::datastructures::Projections::degree ( PolyRef p ) [inline]

**0.15.412.3.4 `disc()`** PolyRef smtrat::cadcells::datastructures::Projections::disc ( PolyRef p ) [inline]

**0.15.412.3.5 `evaluate()`** RAN smtrat::cadcells::datastructures::Projections::evaluate ( const Assignment & sample, IndexedRoot r ) [inline]

**0.15.412.3.6 `factors_nonconst()`** const std::vector<PolyRef>& smtrat::cadcells::datastructures::  
::Projections::factors\_nonconst ( PolyRef p ) [inline]

**0.15.412.3.7 `has_const_coeff()`** bool smtrat::cadcells::datastructures::Projections::has\_const\_coeff ( PolyRef p ) const [inline]

**0.15.412.3.8 `is_const()`** bool smtrat::cadcells::datastructures::Projections::is\_const ( PolyRef p ) [inline]

**0.15.412.3.9 `is_disc_zero()`** bool smtrat::cadcells::datastructures::Projections::is\_disc\_zero ( const Assignment & sample, PolyRef p ) [inline]

**0.15.412.3.10 `is_ldcf_zero()`** bool smtrat::cadcells::datastructures::Projections::is\_ldcf\_zero ( const Assignment & sample, PolyRef p ) [inline]

**0.15.412.3.11 `is_nullified()`** bool smtrat::cadcells::datastructures::Projections::is\_nullified ( const Assignment & sample, PolyRef p ) [inline]

**0.15.412.3.12 `is_zero()` [1/2]** `bool smtrat::cadcells::datastructures::Projections::is_zero (`  
`const Assignment & sample,`  
`PolyRef p ) [inline]`

**0.15.412.3.13 `is_zero()` [2/2]** `bool smtrat::cadcells::datastructures::Projections::is_zero (`  
`PolyRef p ) [inline]`

**0.15.412.3.14 `know_disc()`** `bool smtrat::cadcells::datastructures::Projections::know_disc (`  
`PolyRef p ) const [inline]`

**0.15.412.3.15 `known()`** `bool smtrat::cadcells::datastructures::Projections::known (`  
`const Polynomial & p ) const [inline]`

**0.15.412.3.16 `ldcf()`** `PolyRef smtrat::cadcells::datastructures::Projections::ldcf (`  
`PolyRef p ) [inline]`

**0.15.412.3.17 `main_var()`** `auto smtrat::cadcells::datastructures::Projections::main_var (`  
`PolyRef p ) const [inline]`

**0.15.412.3.18 `num_roots()`** `size_t smtrat::cadcells::datastructures::Projections::num_roots (`  
`const Assignment & sample,`  
`PolyRef p ) [inline]`

**0.15.412.3.19 `polys()` [1/2]** `auto& smtrat::cadcells::datastructures::Projections::polys ( )`  
`[inline]`

**0.15.412.3.20 `polys()` [2/2]** `const auto& smtrat::cadcells::datastructures::Projections::polys ( )`  
`const [inline]`

**0.15.412.3.21 `real_roots()`** `std::vector<RAN> smtrat::cadcells::datastructures::Projections::real_roots (`  
`const Assignment & sample,`  
`PolyRef p ) [inline]`

**0.15.412.3.22 `res()`** `PolyRef smtrat::cadcells::datastructures::Projections::res (`  
`PolyRef p,`  
`PolyRef q ) [inline]`

**0.15.412.3.23 `simplest_nonzero_coeff()`** `PolyRef smtrat::cadcells::datastructures::Projections::simplest_nonzero_coeff (`  
`const Assignment & sample,`  
`PolyRef p,`  
`std::function< bool(const Polynomial &, const Polynomial &) > compare ) const`  
`[inline]`

### 0.15.413 smtrat::cadcells::operators::mccallum\_filtered\_impl::PropertiesSet Struct Reference

```
#include <operator_mccallum_filtered.h>
```

#### Public Types

- using `type = datastructures::PropertiesT< properties::poly_sgn_inv, properties::poly_irreducible_sgn_inv, properties::poly_semi_sgn_inv, properties::poly_irreducible_semi_sgn_inv, properties::poly_ord_inv, properties::root_well_def, properties::poly_pdel, properties::cell_connected, properties::poly_additional_root_outside, properties::poly_ord_inv_base, properties::root_ordering_holds >`

#### 0.15.413.1 Member Typedef Documentation

**0.15.413.1.1 type** using `smtrat::cadcells::operators::mccallum_filtered_impl::PropertiesSet::type = datastructures::PropertiesT<properties::poly_sgn_inv, properties::poly_irreducible_sgn_inv, properties::poly...`

### 0.15.414 smtrat::cadcells::operators::PropertiesSet< Op > Struct Template Reference

### 0.15.415 smtrat::cadcells::operators::PropertiesSet< op::mccallum > Struct Reference

```
#include <operator_mccallum.h>
```

#### Public Types

- using `type = datastructures::PropertiesT< properties::poly_sgn_inv, properties::poly_irreducible_sgn_inv, properties::poly_semi_sgn_inv, properties::poly_ord_inv, properties::root_well_def, properties::poly_pdel, properties::cell_connected >`

#### 0.15.415.1 Member Typedef Documentation

**0.15.415.1.1 type** using `smtrat::cadcells::operators::PropertiesSet< op::mccallum >::type = datastructures::PropertiesT<properties::poly_sgn_inv, properties::poly_irreducible_sgn_inv, properties::poly...`

### 0.15.416 smtrat::cadcells::datastructures::PropertiesT< Ts > Struct Template Reference

Set of properties.

```
#include <properties.h>
```

#### 0.15.416.1 Detailed Description

```
template<typename... Ts>
struct smtrat::cadcells::datastructures::PropertiesT< Ts >
```

Set of properties.

This is a recursive template. The list of template parameters specifies the type of properties which can be hold by this set.

Note that only properties of the same level should be stored within this datastructure.

Properties have the following requirements:

- They need to implement `level()`.
- They need to implement `hash_on_level()`, `operator<` and `operator==` for comparing with properties of the same type and level.

- They need to have a member `static constexpr bool is_flag`. If set to `true`, this indicates that there is only one property of this kind per level and thus, it is not stored in a set but only a Boolean flag is stored.
- They need to implement operator`<<`.

### 0.15.417 `smtrat::cadcells::datastructures::PropertiesT< T, Ts... >` Struct Template Reference

```
#include <properties.h>
```

#### Data Fields

- `PropertiesTContent< T, T::is_flag >::type content`

#### 0.15.417.1 Field Documentation

```
0.15.417.1.1 content template<class T , class... Ts>
PropertiesTContent<T, T::is_flag>::type smtrat::cadcells::datastructures::PropertiesT< T,
Ts... >::content
```

### 0.15.418 `smtrat::cadcells::datastructures::PropertiesTContent< T, is_flag >` Struct Template Reference

### 0.15.419 `smtrat::cadcells::datastructures::PropertiesTContent< T, false >` Struct Template Reference

```
#include <properties.h>
```

#### Public Types

- using `type = PropertiesTSet< T >`

#### 0.15.419.1 Member Typedef Documentation

```
0.15.419.1.1 type template<typename T >
using smtrat::cadcells::datastructures::PropertiesTContent< T, false >::type = PropertiesTSet<T>
```

### 0.15.420 `smtrat::cadcells::datastructures::PropertiesTContent< T, true >` Struct Template Reference

```
#include <properties.h>
```

#### Public Types

- using `type = bool`

#### 0.15.420.1 Member Typedef Documentation

```
0.15.420.1.1 type template<typename T >
using smtrat::cadcells::datastructures::PropertiesTContent< T, true >::type = bool
```

## 0.15.421 smtrat::cadcells::datastructures::property\_hash< T > Struct Template Reference

```
#include <properties.h>
```

### Public Member Functions

- std::size\_t [operator\(\)](#) (const T &p) const

#### 0.15.421.1 Member Function Documentation

```
0.15.421.1.1 operator() template<typename T >
std::size_t smtrat::cadcells::datastructures::property_hash< T >::operator() (
 const T & p) const [inline]
```

## 0.15.422 smtrat::PseudoBoolEncoder Class Reference

Base class for a PseudoBoolean Encoder.

```
#include <PseudoBoolEncoder.h>
```

### Public Member Functions

- std::optional< [FormulaT](#) > [encode](#) (const [ConstraintT](#) &constraint)  
*Encodes an arbitrary constraint.*
- virtual [Rational encodingSize](#) (const [ConstraintT](#) &constraint)
- virtual bool [canEncode](#) (const [ConstraintT](#) &constraint)=0
- virtual std::string [name](#) ()

### Data Fields

- std::size\_t [problem\\_size](#)

### Protected Member Functions

- virtual std::optional< [FormulaT](#) > [doEncode](#) (const [ConstraintT](#) &constraint)=0
- [FormulaT generateVarChain](#) (const std::set< carl::Variable > &vars, carl::FormulaType type)

#### 0.15.422.1 Detailed Description

Base class for a PseudoBoolean Encoder.

It takes a arithmetic constraint and converts it to a boolean Formula

#### 0.15.422.2 Member Function Documentation

```
0.15.422.2.1 canEncode() virtual bool smtrat::PseudoBoolEncoder::canEncode (
 const ConstraintT & constraint) [pure virtual]
```

Implemented in [smtrat::TotalizerEncoder](#), [smtrat::ShortFormulaEncoder](#), [smtrat::RNSEncoder](#), [smtrat::MixedSignEncoder](#), [smtrat::LongFormulaEncoder](#), [smtrat::ExactlyOneCommanderEncoder](#), and [smtrat::CardinalityEncoder](#).

```
0.15.422.2.2 doEncode() virtual std::optional<FormulaT> smtrat::PseudoBoolEncoder::doEncode (
 const ConstraintT & constraint) [protected], [pure virtual]
```

Implemented in [smtrat::TotalizerEncoder](#), [smtrat::ShortFormulaEncoder](#), [smtrat::RNSEncoder](#), [smtrat::MixedSignEncoder](#), [smtrat::LongFormulaEncoder](#), [smtrat::ExactlyOneCommanderEncoder](#), and [smtrat::CardinalityEncoder](#).

```
0.15.422.2.3 encode() std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (
 const ConstraintT & constraint)
```

Encodes an arbitrary constraint.

#### Returns

encoded formula

```
0.15.422.2.4 encodingSize() Rational smtrat::PseudoBoolEncoder::encodingSize (
 const ConstraintT & constraint) [virtual]
```

Reimplemented in [smtrat::TotalizerEncoder](#), [smtrat::ShortFormulaEncoder](#), [smtrat::MixedSignEncoder](#), [smtrat::LongFormulaEncoder](#), [smtrat::ExactlyOneCommanderEncoder](#), and [smtrat::CardinalityEncoder](#).

```
0.15.422.2.5 generateVarChain() FormulaT smtrat::PseudoBoolEncoder::generateVarChain (
 const std::set< carl::Variable > & vars,
 carl::FormulaType type) [protected]
```

```
0.15.422.2.6 name() virtual std::string smtrat::PseudoBoolEncoder::name () [inline], [virtual]
```

Reimplemented in [smtrat::TotalizerEncoder](#), [smtrat::ShortFormulaEncoder](#), [smtrat::MixedSignEncoder](#), [smtrat::LongFormulaEncoder](#), [smtrat::ExactlyOneCommanderEncoder](#), and [smtrat::CardinalityEncoder](#).

## 0.15.422.3 Field Documentation

**0.15.422.3.1 problem\_size** std::size\_t smtrat::PseudoBoolEncoder::problem\_size

## 0.15.423 smtrat::PseudoBoolNormalizer Class Reference

```
#include <PseudoBoolNormalizer.h>
```

### Public Member Functions

- std::pair< std::optional< [FormulaT](#) >, [ConstraintT](#) > **normalize** (const [ConstraintT](#) &constraint)
   
*Modifies the constraint in-place and returns an additional boolean formula.*
- std::map< carl::Variable, carl::Variable > & **substitutedVariables** ()
   
*returns all variable substitutions done by this normalizer instance.*
- [ConstraintT](#) **trim** (const [ConstraintT](#) &constraint)

## 0.15.423.1 Member Function Documentation

```
0.15.423.1.1 normalize() std::pair< std::optional< FormulaT >, ConstraintT > smtrat::PseudoBoolNormalizer::normalize (
 const ConstraintT & constraint)
```

Modifies the constraint in-place and returns an additional boolean formula.

Consider  $-4x \leq 2$

This is equivalent to  $-4*(1 - \text{not } x) \leq 2$  iff.  $-4 + 4 \text{ not } x \leq 2$  iff  $4 \text{ not } x \leq 6$  Since we can not represent this negation in the constraint itself we add a variable y and get  $y \leftrightarrow \text{not } x$  and  $4y \leq 6$

In this particular case we can later on remove y again since the constraint is trivially satisfied.

**0.15.423.1.2 substitutedVariables()** `std::map<carl::Variable, carl::Variable>& smtrat::PseudoBoolNormalizer::substitutedVariables () [inline]`  
returns all variable substitutions done by this normalizer instance.  
An entry {x: y} correlates to the boolean expression y <-> not x  
Useful if all occurrences of the substituted variable should be substituted as well or in general when the correlation is needed.

**0.15.423.1.3 trim()** `ConstraintT smtrat::PseudoBoolNormalizer::trim (const ConstraintT & constraint)`

## 0.15.424 smtrat::parser::QEParser Struct Reference

```
#include <Script.h>
```

### Public Member Functions

- `QEParser (Theories *theories)`
- `carl::Variable resolveVariable (const Identifier &name) const`

### Data Fields

- `Theories * theories`
- `QualifiedIdentifierParser qualifiedidentifier`
- `QuantifierParser quantifier`
- `qi::rule< Iterator, carl::Variable(), Skipper > var`
- `qi::rule< Iterator, qe::QEQuery(), Skipper > main`

### 0.15.424.1 Constructor & Destructor Documentation

**0.15.424.1.1 QEParser()** `smtrat::parser::QEParser::QEParser (Theories * theories) [inline]`

### 0.15.424.2 Member Function Documentation

**0.15.424.2.1 resolveVariable()** `carl::Variable smtrat::parser::QEParser::resolveVariable (const Identifier & name) const [inline]`

### 0.15.424.3 Field Documentation

**0.15.424.3.1 main** `qi::rule<Iterator, qe::QEQuery(), Skipper> smtrat::parser::QEParser::main`

**0.15.424.3.2 qualifiedidentifier** `QualifiedIdentifierParser smtrat::parser::QEParser::qualifiedidentifier`

**0.15.424.3.3 quantifier** `QuantifierParser smtrat::parser::QEParser::quantifier`

**0.15.424.3.4 theories** `Theories* smtrat::parser::QEParser::theories`

---

```
0.15.424.3.5 var qi::rule<Iterator, carl::Variable(), Skipper> smtrat::parser::QEParser::var
```

## 0.15.425 smtrat::parser::QualifiedIdentifierParser Struct Reference

```
#include <Term.h>
```

### Public Member Functions

- **QualifiedIdentifierParser ()**
- **Identifier checkQualification (const Identifier &identifier, const carl::Sort &) const**

### Data Fields

- **IdentifierParser identifier**
- **SortParser sort**
- **qi::rule< Iterator, Identifier(), Skipper > main**

### 0.15.425.1 Constructor & Destructor Documentation

**0.15.425.1.1 QualifiedIdentifierParser()** smtrat::parser::QualifiedIdentifierParser::QualifiedIdentifierParser ( ) [inline]

### 0.15.425.2 Member Function Documentation

**0.15.425.2.1 checkQualification()** Identifier smtrat::parser::QualifiedIdentifierParser::checkQualification ( const Identifier & identifier, const carl::Sort & ) const [inline]

**Todo** Check what can be checked here.

### 0.15.425.3 Field Documentation

**0.15.425.3.1 identifier** IdentifierParser smtrat::parser::QualifiedIdentifierParser::identifier

**0.15.425.3.2 main** qi::rule<**Iterator**, Identifier(), **Skipper**> smtrat::parser::QualifiedIdentifierParser::main

**0.15.425.3.3 sort** SortParser smtrat::parser::QualifiedIdentifierParser::sort

## 0.15.426 smtrat::expression::QuantifierExpression Struct Reference

```
#include <ExpressionContent.h>
```

### Public Member Functions

- **QuantifierExpression (QuantifierType \_type, std::vector< carl::Variable > &&\_variables, Expression &&\_expression)**

**Data Fields**

- `QuantifierType type`
- `std::vector< carl::Variable > variables`
- `Expression expression`

**0.15.426.1 Constructor & Destructor Documentation**

```
0.15.426.1.1 QuantifierExpression() smtrat::expression::QuantifierExpression::QuantifierExpression
(
 QuantifierType _type,
 std::vector< carl::Variable > _variables,
 Expression && _expression) [inline]
```

**0.15.426.2 Field Documentation**

**0.15.426.2.1 expression** `Expression` smtrat::expression::QuantifierExpression::expression

**0.15.426.2.2 type** `QuantifierType` smtrat::expression::QuantifierExpression::type

**0.15.426.2.3 variables** `std::vector<carl::Variable>` smtrat::expression::QuantifierExpression::variables

**0.15.427 smtrat::parser::QuantifierParser Struct Reference**

```
#include <Script.h>
```

**Public Member Functions**

- `QuantifierParser()`

**0.15.427.1 Constructor & Destructor Documentation**

**0.15.427.1.1 QuantifierParser()** smtrat::parser::QuantifierParser::QuantifierParser ( ) [inline]

**0.15.428 Minisat::Queue< T > Class Template Reference**

```
#include <Queue.h>
```

**Public Types**

- `typedef T Key`

**Public Member Functions**

- `Queue()`
- `void clear (bool dealloc=false)`
- `int size () const`
- `const T & operator[] (int index) const`
- `T & operator[] (int index)`

- T `peek () const`
- void `pop ()`
- void `insert (T elem)`

### 0.15.428.1 Member Typedef Documentation

**0.15.428.1.1 Key** template<class T >  
typedef T `Minisat::Queue< T >::Key`

### 0.15.428.2 Constructor & Destructor Documentation

**0.15.428.2.1 Queue()** template<class T >  
`Minisat::Queue< T >::Queue ( ) [inline]`

### 0.15.428.3 Member Function Documentation

**0.15.428.3.1 clear()** template<class T >  
void `Minisat::Queue< T >::clear (`  
    `bool deallocate = false ) [inline]`

**0.15.428.3.2 insert()** template<class T >  
void `Minisat::Queue< T >::insert (`  
    `T elem ) [inline]`

**0.15.428.3.3 operator[]() [1/2]** template<class T >  
T& `Minisat::Queue< T >::operator[] (`  
    `int index ) [inline]`

**0.15.428.3.4 operator[]() [2/2]** template<class T >  
const T& `Minisat::Queue< T >::operator[] (`  
    `int index ) const [inline]`

**0.15.428.3.5 peek()** template<class T >  
T `Minisat::Queue< T >::peek ( ) const [inline]`

**0.15.428.3.6 pop()** template<class T >  
void `Minisat::Queue< T >::pop ( ) [inline]`

**0.15.428.3.7 size()** template<class T >  
int `Minisat::Queue< T >::size ( ) const [inline]`

## 0.15.429 smtrat::mcsat::icp::QueueEntry Struct Reference

```
#include <IntervalPropagation.h>
```

## Data Fields

- double **priority**
- carl::contractor::Contractor< **FormulaT**, **Poly** > **contractor**

### 0.15.429.1 Field Documentation

**0.15.429.1.1 contractor** carl::contractor::Contractor<**FormulaT**, **Poly**> smtrat::mcsat::icp::←  
QueueEntry::contractor

**0.15.429.1.2 priority** double smtrat::mcsat::icp::QueueEntry::priority

## 0.15.430 **benchmax::RandomizationAdaptor< T >** Class Template Reference

Provides iteration over a given std::vector in a pseudo-randomized order.

```
#include <Jobs.h>
```

### Data Structures

- struct **iterator**

### Public Member Functions

- **RandomizationAdaptor** (const std::vector< T > &data)  
*Construct a randomized view on the given vector.*
- auto **begin** () const  
*Return begin of the randomized range.*
- auto **end** () const  
*Return end of the randomized range.*

### 0.15.430.1 Detailed Description

```
template<typename T>
class benchmax::RandomizationAdaptor< T >
```

Provides iteration over a given std::vector in a pseudo-randomized order.

Internally, it computes a factor that is coprime to the number of jobs (and thus generates all indices). We choose the factor in the order of the square root of the number of jobs.

### 0.15.430.2 Constructor & Destructor Documentation

**0.15.430.2.1 RandomizationAdaptor()** template<typename T >
benchmax::RandomizationAdaptor< T >::RandomizationAdaptor (
 const std::vector< T > & data ) [inline]

Construct a randomized view on the given vector.

### 0.15.430.3 Member Function Documentation

**0.15.430.3.1 begin()** template<typename T >
auto benchmax::RandomizationAdaptor< T >::begin ( ) const [inline]

Return begin of the randomized range.

**0.15.430.3.2 end()** template<typename T>  
auto **benchmark::RandomizationAdaptor**< T >::end( ) const [inline]  
Return end of the randomized range.

## 0.15.431 smtrat::RationalCapsule Class Reference

```
#include <RationalCapsule.h>
```

### Public Member Functions

- RationalCapsule (const Rational &aRational, const Rational &bRational, const Rational &cRational)
- const Rational & getARational () const
- const Rational & getBRational () const
- const Rational & getCRational () const

### Data Fields

- Rational aRational = aRational
- Rational bRational = bRational
- Rational cRational = cRational

#### 0.15.431.1 Constructor & Destructor Documentation

**0.15.431.1.1 RationalCapsule()** smtrat::RationalCapsule::RationalCapsule (const Rational & aRational, const Rational & bRational, const Rational & cRational )

#### 0.15.431.2 Member Function Documentation

**0.15.431.2.1 getARational()** const smtrat::Rational & smtrat::RationalCapsule::getARational ( ) const

**0.15.431.2.2 getBRational()** const smtrat::Rational & smtrat::RationalCapsule::getBRational ( ) const

**0.15.431.2.3 getCRational()** const smtrat::Rational & smtrat::RationalCapsule::getCRational ( ) const

#### 0.15.431.3 Field Documentation

**0.15.431.3.1 aRational** Rational smtrat::RationalCapsule::aRational = aRational

**0.15.431.3.2 bRational** Rational smtrat::RationalCapsule::bRational = bRational

**0.15.431.3.3 cRational** Rational smtrat::RationalCapsule::cRational = cRational

## 0.15.432 smtrat::parser::RationalPolicies Struct Reference

Specialization of qi::real\_policies for a Rational.

```
#include <Lexicon.h>
```

### Static Public Member Functions

- template<typename It , typename Attr >  
  static bool [parse\\_nan](#) (It &, It const &, Attr &)
- template<typename It , typename Attr >  
  static bool [parse\\_inf](#) (It &, It const &, Attr &)

### 0.15.432.1 Detailed Description

Specialization of qi::real\_policies for a Rational.

Specifies that neither NaN nor Inf is allowed.

### 0.15.432.2 Member Function Documentation

```
0.15.432.2.1 parse_inf() template<typename It , typename Attr >
static bool smtrat::parser::RationalPolicies::parse_inf (
 It & ,
 It const & ,
 Attr &) [inline], [static]
```

```
0.15.432.2.2 parse_nan() template<typename It , typename Attr >
static bool smtrat::parser::RationalPolicies::parse_nan (
 It & ,
 It const & ,
 Attr &) [inline], [static]
```

## 0.15.433 smtrat::mcsat::onecellcad::RealAlgebraicPoint< Number > Class Template Reference

Represent a multidimensional point whose components are algebraic reals.

```
#include <RealAlgebraicPoint.h>
```

### Public Member Functions

- [RealAlgebraicPoint](#) () noexcept=default  
*Create an empty point of dimension 0.*
- [RealAlgebraicPoint](#) (const std::vector< carl::IntRepRealAlgebraicNumber< Number >> &v)  
*Convert from a vector using its numbers in the same order as components.*
- [RealAlgebraicPoint](#) (std::vector< carl::IntRepRealAlgebraicNumber< Number >> &&v)  
*Convert from a vector using its numbers in the same order as components.*
- [RealAlgebraicPoint](#) (const std::list< carl::IntRepRealAlgebraicNumber< Number >> &v)  
*Convert from a list using its numbers in the same order as components.*
- [RealAlgebraicPoint](#) (const std::initializer\_list< carl::IntRepRealAlgebraicNumber< Number >> &v)  
*Convert from a initializer\_list using its numbers in the same order as components.*
- std::size\_t [dim](#) () const  
*Give the dimension/number of components of this point.*
- [RealAlgebraicPoint](#) [prefixPoint](#) (size\_t componentCount) const  
*Make a (lower dimensional) copy that contains only the first 'componentCount'-many components.*

- `RealAlgebraicPoint conjoin (const carl::IntRepRealAlgebraicNumber< Number > &r)`  
*Create a new point with another given component added at the end of this point, thereby increasing its dimension by 1.*
- `const carl::IntRepRealAlgebraicNumber< Number > & operator[] (std::size_t index) const`  
*Retrieve the component of this point at the given index.*
- `carl::IntRepRealAlgebraicNumber< Number > & operator[] (std::size_t index)`  
*Retrieve the component of this point at the given index.*

### 0.15.433.1 Detailed Description

```
template<typename Number>
class smrat::mcsat::oncellcad::RealAlgebraicPoint< Number >
```

Represent a multidimensional point whose components are algebraic reals.  
This class is just a thin wrapper around vector to have a clearer semantic meaning.

### 0.15.433.2 Constructor & Destructor Documentation

**0.15.433.2.1 RealAlgebraicPoint() [1/5]** template<typename Number >  
`smrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint ( ) [default],`  
`[noexcept]`

Create an empty point of dimension 0.

**0.15.433.2.2 RealAlgebraicPoint() [2/5]** template<typename Number >  
`smrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint (`  
`const std::vector< carl::IntRepRealAlgebraicNumber< Number >> & v ) [inline],`  
`[explicit]`

Convert from a vector using its numbers in the same order as components.

**0.15.433.2.3 RealAlgebraicPoint() [3/5]** template<typename Number >  
`smrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint (`  
`std::vector< carl::IntRepRealAlgebraicNumber< Number >> && v ) [inline], [explicit]`

Convert from a vector using its numbers in the same order as components.

**0.15.433.2.4 RealAlgebraicPoint() [4/5]** template<typename Number >  
`smrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint (`  
`const std::list< carl::IntRepRealAlgebraicNumber< Number >> & v ) [inline],`  
`[explicit]`

Convert from a list using its numbers in the same order as components.

**0.15.433.2.5 RealAlgebraicPoint() [5/5]** template<typename Number >  
`smrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::RealAlgebraicPoint (`  
`const std::initializer_list< carl::IntRepRealAlgebraicNumber< Number >> & v )`  
`[inline]`

Convert from a initializer\_list using its numbers in the same order as components.

### 0.15.433.3 Member Function Documentation

**0.15.433.3.1 `conjoin()`** template<typename Number >  
RealAlgebraicPoint smtrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::conjoin ( const carl::IntRepRealAlgebraicNumber< Number > & r ) [inline]  
Create a new point with another given component added at the end of this point, thereby increasing its dimension by 1.  
The original point remains untouched.

**0.15.433.3.2 `dim()`** template<typename Number >  
std::size\_t smtrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::dim ( ) const [inline]  
Give the dimension/number of components of this point.

**0.15.433.3.3 `operator[]() [1/2]`** template<typename Number >  
carl::IntRepRealAlgebraicNumber<Number>& smtrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::operator[] ( std::size\_t index ) [inline]  
Retrieve the component of this point at the given index.

**0.15.433.3.4 `operator[]() [2/2]`** template<typename Number >  
const carl::IntRepRealAlgebraicNumber<Number>& smtrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::operator[] ( std::size\_t index ) const [inline]  
Retrieve the component of this point at the given index.

**0.15.433.3.5 `prefixPoint()`** template<typename Number >  
RealAlgebraicPoint smtrat::mcsat::oncellcad::RealAlgebraicPoint< Number >::prefixPoint ( size\_t componentCount ) const [inline]  
Make a (lower dimensional) copy that contains only the first 'componentCount'-many components.

## 0.15.434 `smtrat::fixedsize_allocator< T >::rebind< U >` Struct Template Reference

```
#include <alloc.h>
```

### Public Types

- `typedef fixedsize_allocator< U > other`

#### 0.15.434.1 Member TypeDef Documentation

**0.15.434.1.1 `other`** template<typename T >  
template<class U >  
`typedef fixedsize_allocator<U> smtrat::fixedsize_allocator< T >::rebind< U >::other`

## 0.15.435 `smtrat::mcsat::oncellcad::recursive::RecursiveCAD` Class Reference

```
#include <OneCellCAD.h>
```

### Public Member Functions

- `bool isAlreadyProcessed (const TagPoly &poly)`
- `void shrinkSingleComponent (const std::size_t polyLevel, const Poly &poly, CADCell &cell)`

*Shrink the component of the 'cell' at the 'boundCandidate's level around the given point if the 'boundCandidate' is a tighter bound.*

- std::vector< Poly > partialDerivativesLayerWithSomeNonEarlyVanishingPoly (const TagPoly &mainPoly)  
*Find the smallest index m such that in the set S of all m-th partial derivatives of the given poly, such that there is a derivative that does not vanish early [brown15].*
- ShrinkResult refineNull (const TagPoly &boundCandidate, CADCell &cell)
- ShrinkResult shrinkCellWithEarlyVanishingPoly (const TagPoly &boundCandidate, CADCell &cell)
- ShrinkResult shrinkCellWithIrreducibleFactorsOfPoly (const TagPoly &poly, CADCell &cell)
- ShrinkResult refineNonNull (const TagPoly &boundCandidate, CADCell &cell)
- ShrinkResult shrinkCellWithPolyHavingPointAsRoot (const TagPoly &boundCandidate, CADCell &cell)
- bool hasPolyLastVariableRootWithinBounds (const RAN &low, const RAN &high, const Poly &poly, const std::size\_t polyLevel)  
*Check if there is a root of the given polynomial—that becomes univariate after pluggin in all but the last variable wrt.*
- ShrinkResult shrinkCellWithNonRootPoint (const TagPoly &boundCandidate, CADCell &cell)
- ShrinkResult shrinkCell (const TagPoly &boundCandidate, CADCell &cell)  
*Merge the given open OpenCADCell 'cell' that contains the given 'point' (called "alpha" in [brown15]) with a polynomial 'poly'.*
- std::optional< CADCell > pointEnclosingCADCell (const std::vector< std::vector< onecellcad::TagPoly >> &polys)  
*Construct a single CADCell that contains the given 'point' and is sign-invariant for the given polynomials in 'polys'.*
- OneCellCAD (const std::vector< carl::Variable > &variableOrder, const RealAlgebraicPoint< Rational > &point)
- std::map< carl::Variable, RAN > prefixPointToStdMap (const std::size\_t componentCount)  
*Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.*
- std::vector< RAN > isolateLastVariableRoots (const std::size\_t polyLevel, const Poly &poly)  
*src/lib/datastructures/mcsat/onecellcad/OneCellCAD.h Given a poly p(x1,...,xn), return all roots of the univariate poly p(a1,..,an-1, xn), where a1,..,an-1 are the first algebraic real components from 'point' (SORTED!).*
- bool vanishesEarly (const std::size\_t polyLevel, const Poly &poly)  
*Check if an n-variate 'poly' p(x1,...,xn) with n>=1 already becomes the zero poly after plugging in p(a1,..,an-1, xn), where a1,..,an-1 are the first n-1 algebraic real components from 'point'.*
- bool isPointRootOfPoly (const std::size\_t polyLevel, const Poly &poly)
- bool isPointRootOfPoly (const TagPoly &poly)
- std::optional< Poly > coeffNonNull (const TagPoly &boundCandidate)
- bool isMainPointInsideCell (const CADCell &cell)

## Data Fields

- const std::vector< carl::Variable > & variableOrder  
*Variables can also be indexed by level.*
- const RealAlgebraicPoint< Rational > & point

### 0.15.435.1 Member Function Documentation

**0.15.435.1.1 coeffNonNull()** std::optional<Poly> smtrat::mcsat::onecellcad::OneCellCAD::coeffNonNull (const TagPoly & boundCandidate ) [inline], [inherited]

**0.15.435.1.2 hasPolyLastVariableRootWithinBounds()** bool smtrat::mcsat::onecellcad::recursive::RecursiveCAD::hasPolyLastVariableRootWithinBounds (const RAN & low, const RAN & high, const Poly & poly, const std::size\_t polyLevel ) [inline]

Check if there is a root of the given polynomial—that becomes univariate after pluggin in all but the last variable wrt. the given variableOrder—, that lies between the given 'low' and 'high' bounds.

**0.15.435.1.3 isAlreadyProcessed()** `bool smtrat::mcsat::oncellcad::recursive::RecursiveCAD::isAlreadyProcessed ( const TagPoly & poly ) [inline]`

**0.15.435.1.4 isMainPointInsideCell()** `bool smtrat::mcsat::oncellcad::OneCellCAD::isMainPointInsideCell ( const CADCell & cell ) [inline], [inherited]`

**0.15.435.1.5 isolateLastVariableRoots()** `std::vector<RAN> smtrat::mcsat::oncellcad::OneCellCAD::isolateLastVariableRoots ( const std::size_t polyLevel, const Poly & poly ) [inline], [inherited]`  
src/lib/datastructures/mcsat/oncellcad/OneCellCAD.h Given a poly  $p(x_1, \dots, x_n)$ , return all roots of the univariate poly  $p(a_1, \dots, a_{n-1}, x_n)$ , where  $a_1, \dots, a_{n-1}$  are the first algebraic real components from 'point' (SORTED!).

**0.15.435.1.6 isPointRootOfPoly() [1/2]** `bool smtrat::mcsat::oncellcad::OneCellCAD::isPointRootOfPoly ( const std::size_t polyLevel, const Poly & poly ) [inline], [inherited]`

**0.15.435.1.7 isPointRootOfPoly() [2/2]** `bool smtrat::mcsat::oncellcad::OneCellCAD::isPointRootOfPoly ( const TagPoly & poly ) [inline], [inherited]`

**0.15.435.1.8 OneCellCAD()** `smtrat::mcsat::oncellcad::OneCellCAD::OneCellCAD [inline]`

**0.15.435.1.9 partialDerivativesLayerWithSomeNonEarlyVanishingPoly()** `std::vector<Poly> smtrat::mcsat::oncellcad::recursive::RecursiveCAD::partialDerivativesLayerWithSomeNonEarlyVanishingPoly ( const TagPoly & mainPoly ) [inline]`

Find the smallest index  $m$  such that in the set  $S$  of all  $m$ -th partial derivatives of the given poly, such that there is a derivative that does not vanish early [brown15].

Note that  $m > 0$  i.e, this function never just returns the given poly, which is the 0th partial derivative of itself.

#### Parameters

|                   |                                                                              |
|-------------------|------------------------------------------------------------------------------|
| <code>poly</code> | must not be a const-poly like 2, otherwise this function does not terminate. |
|-------------------|------------------------------------------------------------------------------|

#### Returns

Actually only returns the set  $S$

**0.15.435.1.10 pointEnclosingCADCell()** `std::optional<CADCell> smtrat::mcsat::oncellcad::recursive::RecursiveCAD::pointEnclosingCADCell ( const std::vector< std::vector< oncellcad::TagPoly >> & polys ) [inline]`

Construct a single CADCell that contains the given 'point' and is sign-invariant for the given polynomials in 'polys'. The construction fails if 'polys' are not well-oriented [mccallum84]. Note that this cell is cylindrical only with respect to the given 'variableOrder'.

## Parameters

|                      |                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>variableOrder</i> | must contain unique variables and at least one, because constant polynomials (without a variable) are prohibited.   |
| <i>point</i>         | <i>point.size() &gt;= variables.size()</i> .                                                                        |
| <i>polys</i>         | must contain only non-constant, irreducible polynomials that mention only variables that appear in 'variableOrder'. |

**0.15.435.1.11 `prefixPointToStdMap()`** `std::map<carl::Variable, RAN> smtrat::mcsat::oncellcad::prefixPointToStdMap (`

`const std::size_t componentCount ) [inline], [inherited]`

Create a mapping from the first variables (with index 0 to 'componentCount') in the 'variableOrder' to the first components of the given 'point'.

**0.15.435.1.12 `refineNonNull()`** `ShrinkResult smtrat::mcsat::oncellcad::recursive::RecursiveCAD::refineNonNull (`

`const TagPoly & boundCandidate,`  
`CADCell & cell ) [inline]`

**0.15.435.1.13 `refineNull()`** `ShrinkResult smtrat::mcsat::oncellcad::recursive::RecursiveCAD::refineNull (`

`const TagPoly & boundCandidate,`  
`CADCell & cell ) [inline]`

**0.15.435.1.14 `shrinkCell()`** `ShrinkResult smtrat::mcsat::oncellcad::recursive::RecursiveCAD::shrinkCell (`

`const TagPoly & boundCandidate,`  
`CADCell & cell ) [inline]`

Merge the given open OpenCADCell 'cell' that contains the given 'point' (called "alpha" in [brown15]) with a polynomial 'poly'.

Before the merge 'cell' represents a region that is sign-invariant on other (previously merged) polynomials (all signs are non-zero). The returned cell represents a region that is additionally sign-invariant on 'poly' (also with non-zero sign).

## Returns

either an OpenCADCell or nothing (representing a failed construction)

**0.15.435.1.15 `shrinkCellWithEarlyVanishingPoly()`** `ShrinkResult smtrat::mcsat::oncellcad::recursive::RecursiveCAD::shrinkCellWithEarlyVanishingPoly (`

`const TagPoly & boundCandidate,`  
`CADCell & cell ) [inline]`

**0.15.435.1.16 `shrinkCellWithIrreducibleFactorsOfPoly()`** `ShrinkResult smtrat::mcsat::oncellcad::recursive::RecursiveCAD::shrinkCellWithIrreducibleFactorsOfPoly (`

`const TagPoly & poly,`  
`CADCell & cell ) [inline]`

---

**0.15.435.1.17 shrinkCellWithNonRootPoint()** `ShrinkResult smtrat::mcsat::oncellcad::recursive::←  
RecursiveCAD::shrinkCellWithNonRootPoint (const TagPoly & boundCandidate,  
CADCell & cell ) [inline]`

**0.15.435.1.18 shrinkCellWithPolyHavingPointAsRoot()** `ShrinkResult smtrat::mcsat::oncellcad::←  
::recursive::RecursiveCAD::shrinkCellWithPolyHavingPointAsRoot (const TagPoly & boundCandidate,  
CADCell & cell ) [inline]`

**0.15.435.1.19 shrinkSingleComponent()** `void smtrat::mcsat::oncellcad::recursive::RecursiveCAD::←  
::shrinkSingleComponent (const std::size_t polyLevel,  
const Poly & poly,  
CADCell & cell ) [inline]`

Shrink the component of the 'cell' at the 'boundCandidate's level around the given point if the 'boundCandidate' is a tighter bound.

Don't recursively shrink lower level components of the cell. Note that a cell's sector may collapse into a section, which is why we need to pass in a cell and not only its sector. Note that this shrink function is always "successful", even if the cell was not shrunk.

#### Parameters

|                             |                                                                        |
|-----------------------------|------------------------------------------------------------------------|
| <code>boundCandidate</code> | Must be a non-const irreducible polynomial that does not vanish early. |
|-----------------------------|------------------------------------------------------------------------|

**0.15.435.1.20 vanishesEarly()** `bool smtrat::mcsat::oncellcad::OneCellCAD::vanishesEarly (const std::size_t polyLevel,  
const Poly & poly ) [inline], [inherited]`

Check if an n-variate 'poly'  $p(x_1, \dots, x_n)$  with  $n \geq 1$  already becomes the zero poly after plugging in  $p(a_1, \dots, a_{n-1}, x_n)$ , where  $a_1, \dots, a_{n-1}$  are the first  $n-1$  algebraic real components from 'point'.

## 0.15.435.2 Field Documentation

**0.15.435.2.1 point** `const RealAlgebraicPoint<Rational>& smtrat::mcsat::oncellcad::OneCellCAD::point [inherited]`

**0.15.435.2.2 variableOrder** `const std::vector<carl::Variable>& smtrat::mcsat::oncellcad::OneCellCAD::variableOrder [inherited]`

Variables can also be indexed by level.

Polys with mathematical level 1 contain the variable in `variableOrder[0]`

## 0.15.436 smtrat::cad::projection::Reducta< Poly > Struct Template Reference

Construct the set of reducta of the given polynomial.

```
#include <utils.h>
```

#### Public Member Functions

- `Reducta (const Poly &p)`

## Data Fields

- **T elements**

*STL member.*

### 0.15.436.1 Detailed Description

```
template<typename Poly>
struct smtrat::cad::projection::Reducta< Poly >
```

Construct the set of reducta of the given polynomial.  
This only adds a custom constructor to a std::

### 0.15.436.2 Constructor & Destructor Documentation

```
0.15.436.2.1 Reducta() template<typename Poly >
smtrat::cad::projection::Reducta< Poly >::Reducta (
 const Poly & p) [inline]
```

### 0.15.436.3 Field Documentation

```
0.15.436.3.1 elements T std::vector< T >::elements [inherited]
STL member.
```

## 0.15.437 Minisat::RegionAllocator< T > Class Template Reference

```
#include <Alloc.h>
```

### Public Types

- enum { **Ref\_Undef** = UINT32\_MAX }
- enum { **Unit\_Size** = sizeof(uint32\_t) }
- typedef uint32\_t **Ref**

### Public Member Functions

- [\*\*RegionAllocator\*\* \(uint32\\_t start\\_cap=1024 \\*1024\)](#)
- [\*\*~RegionAllocator\*\* \(\)](#)
- [\*\*uint32\\_t size\*\* \(\) const](#)
- [\*\*uint32\\_t wasted\*\* \(\) const](#)
- [\*\*Ref alloc\*\* \(int \*\*size\*\*\)](#)
- [\*\*void free\*\* \(int \*\*size\*\*\)](#)
- [\*\*T & operator\[\]\*\* \(\*\*Ref r\*\*\)](#)
- [\*\*const T & operator\[\]\*\* \(\*\*Ref r\*\*\) const](#)
- [\*\*T \\* lea\*\* \(\*\*Ref r\*\*\)](#)
- [\*\*const T \\* lea\*\* \(\*\*Ref r\*\*\) const](#)
- [\*\*Ref ael\*\* \(const T \\*t\)](#)
- [\*\*void moveTo\*\* \(\*\*RegionAllocator\*\* &to\)](#)

### 0.15.437.1 Member Typedef Documentation

**0.15.437.1.1 Ref** template<class T >  
typedef uint32\_t Minisat::RegionAllocator< T >::Ref

### 0.15.437.2 Member Enumeration Documentation

**0.15.437.2.1 anonymous enum** template<class T >  
anonymous enum

Enumerator

|           |
|-----------|
| Ref_Undef |
|-----------|

**0.15.437.2.2 anonymous enum** template<class T >  
anonymous enum

Enumerator

|           |
|-----------|
| Unit_Size |
|-----------|

### 0.15.437.3 Constructor & Destructor Documentation

**0.15.437.3.1 RegionAllocator()** template<class T >  
Minisat::RegionAllocator< T >::RegionAllocator (   
    uint32\_t start\_cap = 1024 \* 1024 ) [inline], [explicit]

**0.15.437.3.2 ~RegionAllocator()** template<class T >  
Minisat::RegionAllocator< T >::~RegionAllocator ( ) [inline]

### 0.15.437.4 Member Function Documentation

**0.15.437.4.1 ael()** template<class T >  
Ref Minisat::RegionAllocator< T >::ael (   
    const T \* t ) [inline]

**0.15.437.4.2 alloc()** template<class T >  
RegionAllocator< T >::Ref Minisat::RegionAllocator< T >::alloc (   
    int size )

**0.15.437.4.3 free()** template<class T >  
void Minisat::RegionAllocator< T >::free (   
    int size ) [inline]

**0.15.437.4.4 lea() [1/2]** template<class T >  
T\* Minisat::RegionAllocator< T >::lea (  
    Ref r ) [inline]

**0.15.437.4.5 lea() [2/2]** template<class T >  
const T\* Minisat::RegionAllocator< T >::lea (  
    Ref r ) const [inline]

**0.15.437.4.6 moveTo()** template<class T >  
void Minisat::RegionAllocator< T >::moveTo (  
    RegionAllocator< T > & to ) [inline]

**0.15.437.4.7 operator[]() [1/2]** template<class T >  
T& Minisat::RegionAllocator< T >::operator[] (  
    Ref r ) [inline]

**0.15.437.4.8 operator[]() [2/2]** template<class T >  
const T& Minisat::RegionAllocator< T >::operator[] (  
    Ref r ) const [inline]

**0.15.437.4.9 size()** template<class T >  
uint32\_t Minisat::RegionAllocator< T >::size ( ) const [inline]

**0.15.437.4.10 wasted()** template<class T >  
uint32\_t Minisat::RegionAllocator< T >::wasted ( ) const [inline]

## 0.15.438 smtrat::NNFRewriter::remove\_xor\_first\_arg Struct Reference

#include <NNFRewriter.h>

### Public Member Functions

- `remove_xor_first_arg (NNFRewriter &r_, const FormulaT &f_)`
- `~remove_xor_first_arg ()`

### 0.15.438.1 Constructor & Destructor Documentation

**0.15.438.1.1 remove\_xor\_first\_arg()** smtrat::NNFRewriter::remove\_xor\_first\_arg::remove\_xor\_<br>
first\_arg (  
    NNFRewriter & r\_,  
    const FormulaT & f\_ ) [inline]

**0.15.438.1.2 ~remove\_xor\_first\_arg()** smtrat::NNFRewriter::remove\_xor\_first\_arg::~remove\_xor\_<br>
first\_arg ( ) [inline]

## 0.15.439 smtrat::ReplaceVariablesRewriter Struct Reference

#include <ReplaceVariablesRewriter.h>

## Public Types

- `typedef carl::UTerm term_type`
- `typedef carl::UEquality UEquality`
- `typedef carl::UVariable UVariable`
- `typedef carl::UninterpretedFunction UninterpretedFunction`
- `typedef carl::UFInstance UFInstance`

## Public Member Functions

- `ReplaceVariablesRewriter (std::unordered_map< FormulaT, bool > &facts, ModuleWrapper< EQModule< EQSettingsForPreprocessing >> &helper)`
- `FormulaT rewrite_ueq (const FormulaT &ueq)`
- `void enter_default (const FormulaT &)`
- `FormulaT rewrite_default (const FormulaT &, const FormulaT &result)`
- `void enter_and (const FormulaT &)`
- `FormulaT rewrite_and (const FormulaT &, FormulaSetT &&subformulas)`

### 0.15.439.1 Member Typedef Documentation

**0.15.439.1.1 `term_type`** `typedef carl::UTerm smtrat::ReplaceVariablesRewriter::term_type`

**0.15.439.1.2 `UEquality`** `typedef carl::UEquality smtrat::ReplaceVariablesRewriter::UEquality`

**0.15.439.1.3 `UFInstance`** `typedef carl::UFInstance smtrat::ReplaceVariablesRewriter::UFInstance`

**0.15.439.1.4 `UninterpretedFunction`** `typedef carl::UninterpretedFunction smtrat::ReplaceVariablesRewriter::Uninterpret`

**0.15.439.1.5 `UVariable`** `typedef carl::UVariable smtrat::ReplaceVariablesRewriter::UVariable`

### 0.15.439.2 Constructor & Destructor Documentation

**0.15.439.2.1 `ReplaceVariablesRewriter()`** `smtrat::ReplaceVariablesRewriter::ReplaceVariablesRewriter ( std::unordered_map< FormulaT, bool > & facts, ModuleWrapper< EQModule< EQSettingsForPreprocessing >> & helper ) [inline]`

### 0.15.439.3 Member Function Documentation

**0.15.439.3.1 `enter_and()`** `void smtrat::ReplaceVariablesRewriter::enter_and ( const FormulaT & ) [inline]`

**0.15.439.3.2 `enter_default()`** `void smtrat::ReplaceVariablesRewriter::enter_default ( const FormulaT & ) [inline]`

```
0.15.439.3.3 rewrite_and() FormulaT smrat::ReplaceVariablesRewriter::rewrite_and (
 const FormulaT &,
 FormulaSetT && subformulas) [inline]
```

```
0.15.439.3.4 rewrite_default() FormulaT smrat::ReplaceVariablesRewriter::rewrite_default (
 const FormulaT &,
 const FormulaT & result) [inline]
```

```
0.15.439.3.5 rewrite_ueq() FormulaT smrat::ReplaceVariablesRewriter::rewrite_ueq (
 const FormulaT & ueq) [inline]
```

## 0.15.440 **delta::ErrorHandler::result< typename >** Struct Template Reference

```
#include <Parser.h>
```

### Public Types

- `typedef qi::error_handler_result type`

#### 0.15.440.1 Member TypeDef Documentation

```
0.15.440.1.1 type template<typename >
typedef qi::error_handler_result delta::ErrorHandler::result< typename >::type
```

## 0.15.441 **smrat::parser::ErrorHandler::result< typename >** Struct Template Reference

```
#include <Script.h>
```

### Public Types

- `typedef qi::error_handler_result type`

#### 0.15.441.1 Member TypeDef Documentation

```
0.15.441.1.1 type template<typename >
typedef qi::error_handler_result smrat::parser::ErrorHandler::result< typename >::type
```

## 0.15.442 **smrat::cad::preprocessor::ResultantRule** Class Reference

```
#include <CADPreprocessor.h>
```

### Public Member Functions

- `ResultantRule (Origins &origins, const std::vector< carl::Variable > &vars)`
- `std::optional< std::vector< FormulaT > > compute (const std::map< ConstraintT, ConstraintT > &constraints)`
- `const auto & getNewECs () const`

#### 0.15.442.1 Constructor & Destructor Documentation

```
0.15.442.1.1 ResultantRule() smtrat::cad::preprocessor::ResultantRule::ResultantRule (
 Origins & origins,
 const std::vector< carl::Variable > & vars) [inline]
```

## 0.15.442.2 Member Function Documentation

```
0.15.442.2.1 compute() std::optional< std::vector< FormulaT > > smtrat::cad::preprocessor::←
ResultantRule::compute (
 const std::map< ConstraintT, ConstraintT > & constraints)
```

```
0.15.442.2.2 getNewECs() const auto& smtrat::cad::preprocessor::ResultantRule::getNewECs ()
const [inline]
```

## 0.15.443 benchmax::Results Class Reference

Stores results for for whole benchmax run.

```
#include <Results.h>
```

### Public Member Functions

- std::optional< std::reference\_wrapper< const BenchmarkResult > > **get** (const Tool \*tool, const fs::path &file) const
- const auto & **data** () const
- void **addResult** (const Tool \*tool, const fs::path &file, BenchmarkResult &&results)  
*Add a new result.*
- void **store** (Database &db)  
*Store all results to some database.*
- void **store** (XMLWriter &xml, const Jobs &jobs) const  
*Store all results to a xml file.*

### 0.15.443.1 Detailed Description

Stores results for for whole benchmax run.

Allows for (concurrent) insertion of the individual results via **addResult()**. Eventually results can be stored to a database (if enabled) or to a xml file.

## 0.15.443.2 Member Function Documentation

```
0.15.443.2.1 addResult() void benchmax::Results::addResult (
 const Tool * tool,
 const fs::path & file,
 BenchmarkResult && results) [inline]
```

Add a new result.

```
0.15.443.2.2 data() const auto& benchmax::Results::data () const [inline]
```

```
0.15.443.2.3 get() std::optional<std::reference_wrapper<const BenchmarkResult>> benchmax::←
Results::get (
 const Tool * tool,
 const fs::path & file) const [inline]
```

**0.15.443.2.4 `store()` [1/2]** void benchmax::Results::store ( Database & db ) [inline]  
Store all results to some database.

**0.15.443.2.5 `store()` [2/2]** void benchmax::Results::store ( XMLWriter & xml, const Jobs & jobs ) const [inline]  
Store all results to a xml file.

## 0.15.444 smtrat::RNSEncoder Class Reference

```
#include <RNSEncoder.h>
```

### Public Member Functions

- `RNSEncoder ()`
- bool `canEncode (const ConstraintT &constraint)`
- std::optional< `FormulaT` > `encode (const ConstraintT &constraint)`  
*Encodes an arbitrary constraint.*
- virtual Rational `encodingSize (const ConstraintT &constraint)`
- virtual std::string `name ()`

### Data Fields

- std::size\_t `problem_size`

### Protected Member Functions

- std::optional< `FormulaT` > `doEncode (const ConstraintT &constraint)`
- `FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)`

#### 0.15.444.1 Constructor & Destructor Documentation

**0.15.444.1.1 `RNSEncoder()`** smtrat::RNSEncoder::RNSEncoder ( ) [inline]

#### 0.15.444.2 Member Function Documentation

**0.15.444.2.1 `canEncode()`** bool smtrat::RNSEncoder::canEncode ( const ConstraintT & constraint ) [virtual]  
Implements `smtrat::PseudoBoolEncoder`.

**0.15.444.2.2 `doEncode()`** std::optional< `FormulaT` > smtrat::RNSEncoder::doEncode ( const ConstraintT & constraint ) [protected], [virtual]  
Implements `smtrat::PseudoBoolEncoder`.

**0.15.444.2.3 encode()** `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (`  
    `const ConstraintT & constraint ) [inherited]`

Encodes an arbitrary constraint.

#### Returns

encoded formula

**0.15.444.2.4 encodingSize()** `Rational smtrat::PseudoBoolEncoder::encodingSize (`  
    `const ConstraintT & constraint ) [virtual], [inherited]`

Reimplemented in [smtrat::TotalizerEncoder](#), [smtrat::ShortFormulaEncoder](#), [smtrat::MixedSignEncoder](#), [smtrat::LongFormulaEncoder](#), [smtrat::ExactlyOneCommanderEncoder](#), and [smtrat::CardinalityEncoder](#).

**0.15.444.2.5 generateVarChain()** `FormulaT smtrat::PseudoBoolEncoder::generateVarChain (`  
    `const std::set< carl::Variable > & vars,`  
    `carl::FormulaType type ) [protected], [inherited]`

**0.15.444.2.6 name()** `virtual std::string smtrat::PseudoBoolEncoder::name ( ) [inline], [virtual], [inherited]`

Reimplemented in [smtrat::TotalizerEncoder](#), [smtrat::ShortFormulaEncoder](#), [smtrat::MixedSignEncoder](#), [smtrat::LongFormulaEncoder](#), [smtrat::ExactlyOneCommanderEncoder](#), and [smtrat::CardinalityEncoder](#).

### 0.15.444.3 Field Documentation

**0.15.444.3.1 problem\_size** `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

## 0.15.445 smtrat::cadcells::operators::properties::root\_ordering\_holds Struct Reference

#include <properties.h>

#### Public Member Functions

- `size_t level () const`
- `std::size_t hash_on_level () const`

#### Data Fields

- `datastructures::IndexedRootOrdering ordering`
- `std::size_t lvl`

#### Static Public Attributes

- `static constexpr bool is_flag = false`

### 0.15.445.1 Member Function Documentation

**0.15.445.1.1 hash\_on\_level()** `std::size_t smtrat::cadcells::operators::properties::root_ordering_holds::hash_on_level ( ) const [inline]`

**0.15.445.1.2 level()** size\_t smtrat::cadcells::operators::properties::root\_ordering\_holds::level()  
() const [inline]

## 0.15.445.2 Field Documentation

**0.15.445.2.1 is\_flag** constexpr bool smtrat::cadcells::operators::properties::root\_ordering\_holds::is\_flag = false [static], [constexpr]

**0.15.445.2.2 lvl** std::size\_t smtrat::cadcells::operators::properties::root\_ordering\_holds::lvl

**0.15.445.2.3 ordering** [datastructures::IndexedRootOrdering](#) smtrat::cadcells::operators::properties::root\_ordering\_holds::ordering

## 0.15.446 smtrat::cadcells::operators::properties::root\_well\_def Struct Reference

#include <properties.h>

### Public Member Functions

- size\_t [level](#)() const
- std::size\_t [hash\\_on\\_level](#)() const

### Data Fields

- [datastructures::IndexedRoot](#) root

### Static Public Attributes

- static constexpr bool [is\\_flag](#) = false

## 0.15.446.1 Member Function Documentation

**0.15.446.1.1 hash\_on\_level()** std::size\_t smtrat::cadcells::operators::properties::root\_well\_def::hash\_on\_level()  
() const [inline]

**0.15.446.1.2 level()** size\_t smtrat::cadcells::operators::properties::root\_well\_def::level()  
() const [inline]

## 0.15.446.2 Field Documentation

**0.15.446.2.1 is\_flag** constexpr bool smtrat::cadcells::operators::properties::root\_well\_def::is\_flag = false [static], [constexpr]

**0.15.446.2.2 root** [datastructures::IndexedRoot](#) smtrat::cadcells::operators::properties::root\_well\_def::root

## 0.15.447 smtrat::cadcells::datastructures::RootFunction Class Reference

```
#include <roots.h>
```

### Public Member Functions

- `RootFunction (IndexedRoot data)`
- `RootFunction (CompoundMin &&data)`
- `RootFunction (CompoundMax &&data)`
- `bool is_root () const`
- `bool is_cmin () const`
- `bool is_cmax () const`
- `const IndexedRoot & root () const`
- `const CompoundMin & cmin () const`
- `const CompoundMax & cmax () const`
- `const auto & roots () const`
- `void polys (boost::container::flat_set< PolyRef > &result) const`
- `boost::container::flat_set< PolyRef > polys () const`
- `bool has_poly (const PolyRef poly) const`
- `std::optional< IndexedRoot > poly_root (const PolyRef poly) const`

### Friends

- `bool operator== (const RootFunction &lhs, const RootFunction &rhs)`
- `bool operator< (const RootFunction &lhs, const RootFunction &rhs)`
- `std::ostream & operator<< (std::ostream &os, const RootFunction &data)`

### 0.15.447.1 Constructor & Destructor Documentation

**0.15.447.1.1 RootFunction() [1/3]** smtrat::cadcells::datastructures::RootFunction  
(  
    `IndexedRoot data` )    [inline]

**0.15.447.1.2 RootFunction() [2/3]** smtrat::cadcells::datastructures::RootFunction::RootFunction  
(  
    `CompoundMin && data` )    [inline]

**0.15.447.1.3 RootFunction() [3/3]** smtrat::cadcells::datastructures::RootFunction::RootFunction  
(  
    `CompoundMax && data` )    [inline]

### 0.15.447.2 Member Function Documentation

**0.15.447.2.1 cmax()** const `CompoundMax&` smtrat::cadcells::datastructures::RootFunction::cmax  
( ) const    [inline]

**0.15.447.2.2 cmin()** const `CompoundMin&` smtrat::cadcells::datastructures::RootFunction::cmin  
( ) const    [inline]

**0.15.447.2.3 `has_poly()`** `bool smtrat::cadcells::datastructures::RootFunction::has_poly ( const PolyRef poly ) const [inline]`

**0.15.447.2.4 `is_cmax()`** `bool smtrat::cadcells::datastructures::RootFunction::is_cmax ( ) const [inline]`

**0.15.447.2.5 `is_cmin()`** `bool smtrat::cadcells::datastructures::RootFunction::is_cmin ( ) const [inline]`

**0.15.447.2.6 `is_root()`** `bool smtrat::cadcells::datastructures::RootFunction::is_root ( ) const [inline]`

**0.15.447.2.7 `poly_root()`** `std::optional<IndexedRoot> smtrat::cadcells::datastructures::RootFunction::poly_root ( const PolyRef poly ) const [inline]`

**0.15.447.2.8 `polys() [1/2]`** `boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures::RootFunction::polys ( ) const [inline]`

**0.15.447.2.9 `polys() [2/2]`** `void smtrat::cadcells::datastructures::RootFunction::polys ( boost::container::flat_set<PolyRef> & result ) const [inline]`

**0.15.447.2.10 `root()`** `const IndexedRoot& smtrat::cadcells::datastructures::RootFunction::root ( ) const [inline]`

**0.15.447.2.11 `roots()`** `const auto& smtrat::cadcells::datastructures::RootFunction::roots ( ) const [inline]`

### 0.15.447.3 Friends And Related Function Documentation

**0.15.447.3.1 `operator<`** `bool operator< ( const RootFunction & lhs, const RootFunction & rhs ) [friend]`

**0.15.447.3.2 `operator<<`** `std::ostream& operator<< ( std::ostream & os, const RootFunction & data ) [friend]`

**0.15.447.3.3 `operator==`** `bool operator== ( const RootFunction & lhs, const RootFunction & rhs ) [friend]`

## 0.15.448 `smtrat::mcsat::arithmetic::RootIndexer< RANT >` Class Template Reference

```
#include <RootIndexer.h>
```

### Public Member Functions

- void `add` (const std::vector< RANT > &list)
- void `process` ()
- std::size\_t `size` () const
- std::size\_t `operator[]` (const RANT &ran) const
- const RANT & `operator[]` (std::size\_t n) const
- const RANT & `sampleFrom` (std::size\_t n) const
- bool `is_root` (std::size\_t n) const

#### 0.15.448.1 Member Function Documentation

##### 0.15.448.1.1 `add()` template<typename RANT >

```
void smtrat::mcsat::arithmetic::RootIndexer< RANT >::add (
 const std::vector< RANT > & list) [inline]
```

##### 0.15.448.1.2 `is_root()` template<typename RANT >

```
bool smtrat::mcsat::arithmetic::RootIndexer< RANT >::is_root (
 std::size_t n) const [inline]
```

##### 0.15.448.1.3 `operator[]() [1/2]` template<typename RANT >

```
std::size_t smtrat::mcsat::arithmetic::RootIndexer< RANT >::operator[] (
 const RANT & ran) const [inline]
```

##### 0.15.448.1.4 `operator[]() [2/2]` template<typename RANT >

```
const RANT& smtrat::mcsat::arithmetic::RootIndexer< RANT >::operator[] (
 std::size_t n) const [inline]
```

##### 0.15.448.1.5 `process()` template<typename RANT >

```
void smtrat::mcsat::arithmetic::RootIndexer< RANT >::process () [inline]
```

##### 0.15.448.1.6 `sampleFrom()` template<typename RANT >

```
const RANT& smtrat::mcsat::arithmetic::RootIndexer< RANT >::sampleFrom (
 std::size_t n) const [inline]
```

##### 0.15.448.1.7 `size()` template<typename RANT >

```
std::size_t smtrat::mcsat::arithmetic::RootIndexer< RANT >::size () const [inline]
```

## 0.15.449 `smtrat::cad::Sample` Class Reference

```
#include <Sample.h>
```

## Public Member Functions

- `Sample (const RAN &value)`
- `Sample (const RAN &value, bool isRoot)`
- `Sample (const RAN &value, std::size_t id)`
- `const RAN & value () const`
- `bool isRoot () const`
- `void setIsRoot (bool isRoot)`
- `const SampleLiftedWith & liftedWith () const`
- `SampleLiftedWith & liftedWith ()`
- `const SampleRootOf & rootOf () const`
- `SampleRootOf & rootOf ()`
- `const carl::Bitset & evaluatedWith () const`
- `carl::Bitset & evaluatedWith ()`
- `const carl::Bitset & evaluationResult () const`
- `carl::Bitset & evaluationResult ()`
- `auto getConflictingConstraints () const`
- `bool hasConflictWithConstraint () const`
- `void merge (const Sample &s)`
- `bool operator< (const Sample &s) const`
- `bool operator> (const Sample &s) const`
- `bool operator== (const Sample &s) const`

## Friends

- `std::ostream & operator<< (std::ostream &os, const Sample &s)`

### 0.15.449.1 Constructor & Destructor Documentation

**0.15.449.1.1 `Sample()` [1/3]** smtrat::cad::Sample::Sample (  
  `const RAN & value`) [inline], [explicit]

**0.15.449.1.2 `Sample()` [2/3]** smtrat::cad::Sample::Sample (  
  `const RAN & value,`  
  `bool isRoot`) [inline], [explicit]

**0.15.449.1.3 `Sample()` [3/3]** smtrat::cad::Sample::Sample (  
  `const RAN & value,`  
  `std::size_t id`) [inline], [explicit]

### 0.15.449.2 Member Function Documentation

**0.15.449.2.1 `evaluatedWith()` [1/2]** carl::Bitset& smtrat::cad::Sample::evaluatedWith () [inline]

**0.15.449.2.2 `evaluatedWith()` [2/2]** const carl::Bitset& smtrat::cad::Sample::evaluatedWith ()  
const [inline]

**0.15.449.2.3 evaluationResult() [1/2]** `carl::Bitset& smtrat::cad::Sample::evaluationResult ( )`  
[inline]

**0.15.449.2.4 evaluationResult() [2/2]** `const carl::Bitset& smtrat::cad::Sample::evaluationResult ( ) const` [inline]

**0.15.449.2.5 getConflictingConstraints()** `auto smtrat::cad::Sample::getConflictingConstraints ( ) const` [inline]

**0.15.449.2.6 hasConflictWithConstraint()** `bool smtrat::cad::Sample::hasConflictWithConstraint ( ) const` [inline]

**0.15.449.2.7 isRoot()** `bool smtrat::cad::Sample::isRoot ( ) const` [inline]

**0.15.449.2.8 liftedWith() [1/2]** `SampleLiftedWith& smtrat::cad::Sample::liftedWith ( )` [inline]

**0.15.449.2.9 liftedWith() [2/2]** `const SampleLiftedWith& smtrat::cad::Sample::liftedWith ( ) const` [inline]

**0.15.449.2.10 merge()** `void smtrat::cad::Sample::merge ( const Sample & s )` [inline]

**0.15.449.2.11 operator<()** `bool smtrat::cad::Sample::operator< ( const Sample & s ) const` [inline]

**0.15.449.2.12 operator==()** `bool smtrat::cad::Sample::operator== ( const Sample & s ) const` [inline]

**0.15.449.2.13 operator>()** `bool smtrat::cad::Sample::operator> ( const Sample & s ) const` [inline]

**0.15.449.2.14 rootOf() [1/2]** `SampleRootOf& smtrat::cad::Sample::rootOf ( )` [inline]

**0.15.449.2.15 rootOf() [2/2]** `const SampleRootOf& smtrat::cad::Sample::rootOf ( ) const` [inline]

**0.15.449.2.16 setIsRoot()** `void smtrat::cad::Sample::setIsRoot ( bool isRoot )` [inline]

**0.15.449.2.17 value()** `const RAN& smtrat::cad::Sample::value ( ) const` [inline]

**0.15.449.3 Friends And Related Function Documentation**

**0.15.449.3.1 operator<<** std::ostream& operator<< ( std::ostream & os, const Sample & s ) [friend]

**0.15.450 smtrat::cad::sample\_compare::SampleComparator< Iterator, Strategy > Struct Template Reference**

```
#include <SampleComparator.h>
```

**0.15.451 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::LS > Struct Template Reference**

```
#include <SampleComparator.h>
```

**Public Member Functions**

- bool [operator\(\)](#) (const Iterator &lhs, const Iterator &rhs) const

**0.15.451.1 Member Function Documentation**

**0.15.451.1.1 operator()()** bool smtrat::cad::sample\_compare::SampleComparator\_impl< Iterator , Args >::operator() ( const Iterator & lhs, const Iterator & rhs ) const [inline], [inherited]

**0.15.452 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::LT > Struct Template Reference**

```
#include <SampleComparator.h>
```

**Public Member Functions**

- bool [operator\(\)](#) (const Iterator &lhs, const Iterator &rhs) const

**0.15.452.1 Member Function Documentation**

**0.15.452.1.1 operator()()** bool smtrat::cad::sample\_compare::SampleComparator\_impl< Iterator , Args >::operator() ( const Iterator & lhs, const Iterator & rhs ) const [inline], [inherited]

**0.15.453 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::LTA > Struct Template Reference**

```
#include <SampleComparator.h>
```

**Public Member Functions**

- bool [operator\(\)](#) (const Iterator &lhs, const Iterator &rhs) const

**0.15.453.1 Member Function Documentation**

**0.15.453.1.1 operator()()** bool smtrat::cad::sample\_compare::SampleComparator\_Impl< Iterator ,  
Args >::operator() (  
    const Iterator & lhs,  
    const Iterator & rhs ) const [inline], [inherited]

**0.15.454 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::LTS > Struct Template Reference**

```
#include <SampleComparator.h>
```

**Public Member Functions**

- bool **operator()** (const Iterator &lhs, const Iterator &rhs) const

**0.15.454.1 Member Function Documentation**

**0.15.454.1.1 operator()()** bool smtrat::cad::sample\_compare::SampleComparator\_Impl< Iterator ,  
Args >::operator() (  
    const Iterator & lhs,  
    const Iterator & rhs ) const [inline], [inherited]

**0.15.455 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::LTSA > Struct Template Reference**

```
#include <SampleComparator.h>
```

**Public Member Functions**

- bool **operator()** (const Iterator &lhs, const Iterator &rhs) const

**0.15.455.1 Member Function Documentation**

**0.15.455.1.1 operator()()** bool smtrat::cad::sample\_compare::SampleComparator\_Impl< Iterator ,  
Args >::operator() (  
    const Iterator & lhs,  
    const Iterator & rhs ) const [inline], [inherited]

**0.15.456 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::S > Struct Template Reference**

```
#include <SampleComparator.h>
```

**Public Member Functions**

- bool **operator()** (const Iterator &lhs, const Iterator &rhs) const

**0.15.456.1 Member Function Documentation**

```
0.15.456.1.1 operator() bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
 const It & lhs,
 const It & rhs) const [inline], [inherited]
```

## 0.15.457 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::T > Struct Template Reference

```
#include <SampleComparator.h>
```

### Public Member Functions

- bool [operator\(\)](#) (const Iterator &lhs, const Iterator &rhs) const

#### 0.15.457.1 Member Function Documentation

```
0.15.457.1.1 operator() bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
 const Iterator & lhs,
 const Iterator & rhs) const [inline], [inherited]
```

## 0.15.458 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::TLSA > Struct Template Reference

```
#include <SampleComparator.h>
```

### Public Member Functions

- bool [operator\(\)](#) (const Iterator &lhs, const Iterator &rhs) const

#### 0.15.458.1 Member Function Documentation

```
0.15.458.1.1 operator() bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
 const Iterator & lhs,
 const Iterator & rhs) const [inline], [inherited]
```

## 0.15.459 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::TS > Struct Template Reference

```
#include <SampleComparator.h>
```

### Public Member Functions

- bool [operator\(\)](#) (const Iterator &lhs, const Iterator &rhs) const

#### 0.15.459.1 Member Function Documentation

```
0.15.459.1.1 operator() bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
 const Iterator & lhs,
 const Iterator & rhs) const [inline], [inherited]
```

## 0.15.460 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::TSA > Struct Template Reference

```
#include <SampleComparator.h>
```

### Public Member Functions

- bool [operator\(\)](#) (const Iterator &lhs, const Iterator &rhs) const

#### 0.15.460.1 Member Function Documentation

```
0.15.460.1.1 operator() bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator ,
Args >::operator() (
 const Iterator & lhs,
 const Iterator & rhs) const [inline], [inherited]
```

## 0.15.461 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::Type > Struct Template Reference

```
#include <SampleComparator.h>
```

### Public Member Functions

- bool [operator\(\)](#) (const Iterator &lhs, const Iterator &rhs) const
- bool [reference](#) (const Iterator &lhs, const Iterator &rhs) const
- bool [compare](#) (const Iterator &lhs, const Iterator &rhs) const

#### 0.15.461.1 Member Function Documentation

```
0.15.461.1.1 compare() template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↔
::compare (
 const Iterator & lhs,
 const Iterator & rhs) const [inline]
```

```
0.15.461.1.2 operator() template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↔
::operator() (
 const Iterator & lhs,
 const Iterator & rhs) const [inline]
```

```
0.15.461.1.3 reference() template<typename Iterator >
bool smtrat::cad::sample_compare::SampleComparator< Iterator, SampleCompareStrategy::Type >↔
::reference (
 const Iterator & lhs,
 const Iterator & rhs) const [inline]
```

## 0.15.462 smtrat::cad::sample\_compare::SampleComparator< Iterator, SampleCompareStrategy::Value > Struct Template Reference

```
#include <SampleComparator.h>
```

## Public Member Functions

- bool `operator()` (const Iterator &lhs, const Iterator &rhs) const

### 0.15.462.1 Member Function Documentation

**0.15.462.1.1 `operator()`** `bool smtrat::cad::sample_compare::SampleComparator_impl< Iterator , Args >::operator() (`  
`const Iterator & lhs,`  
`const Iterator & rhs ) const [inline], [inherited]`

### 0.15.463 `smtrat::cad::sample_compare::SampleComparator_impl< It, Args >` Struct Template Reference

```
#include <SampleComparator.h>
```

## Public Member Functions

- bool `operator()` (const It &lhs, const It &rhs) const

### 0.15.463.1 Member Function Documentation

**0.15.463.1.1 `operator()`** `template<typename It , typename... Args>`  
`bool smtrat::cad::sample_compare::SampleComparator_impl< It, Args >::operator() (`  
`const It & lhs,`  
`const It & rhs ) const [inline]`

### 0.15.464 `smtrat::cadcells::datastructures::SampledDerivation< Properties >` Class Template Reference

A `SampledDerivation` is a `DelineatedDerivation` with a sample and an `DelineationInterval` w.r.t.

```
#include <derivation.h>
```

## Public Member Functions

- `SampledDerivation` (`DelineatedDerivationRef< Properties > base, RAN main_sample`)
- `DelineatedDerivationRef< Properties > & delineated ()`
- `const DelineatedDerivationRef< Properties > & delineated () const`
- `const DelineationInterval & cell () const`
- `void delineate_cell ()`

*Determines the cell w.r.t.*

  - `const Assignment & sample () const`
  - `const RAN & main_var_sample () const`
  - `BaseDerivationRef< Properties > & base ()`
  - `const BaseDerivationRef< Properties > & base () const`
  - `Delineation & delin ()`
  - `const Assignment & underlying_sample () const`
  - `DerivationRef< Properties > & underlying ()`
  - `PolyPool & polys ()`
  - `Projections & proj ()`
  - `carl::Variable main_var () const`
  - `size_t level () const`

- template<typename P >  
void **insert** (P property)
- template<typename P >  
bool **contains** (const P &property) const
- template<typename P >  
const **PropertiesTSet**< P > & **properties** () const
- void **merge\_with** (const **SampledDerivation**< Properties > &other)

#### 0.15.464.1 Detailed Description

```
template<typename Properties>
class smrat::cadcells::datastructures::SampledDerivation< Properties >
```

A **SampledDerivation** is a **DelineatedDerivation** with a sample and an **DelineationInterval** w.r.t. to this sample.

##### Template Parameters

|                   |                   |
|-------------------|-------------------|
| <i>Properties</i> | Set of properties |
|-------------------|-------------------|

#### 0.15.464.2 Constructor & Destructor Documentation

```
0.15.464.2.1 SampledDerivation() template<typename Properties >
smrat::cadcells::datastructures::SampledDerivation< Properties >::SampledDerivation (
 DelineatedDerivationRef< Properties > base,
 RAN main_sample) [inline]
```

#### 0.15.464.3 Member Function Documentation

```
0.15.464.3.1 base() [1/2] template<typename Properties >
BaseDerivationRef<Properties>& smrat::cadcells::datastructures::SampledDerivation< Properties
>::base () [inline]
```

```
0.15.464.3.2 base() [2/2] template<typename Properties >
const BaseDerivationRef<Properties>& smrat::cadcells::datastructures::SampledDerivation<
Properties >::base () const [inline]
```

```
0.15.464.3.3 cell() template<typename Properties >
const DelineationInterval& smrat::cadcells::datastructures::SampledDerivation< Properties >::cell
() const [inline]
```

```
0.15.464.3.4 contains() template<typename Properties >
template<typename P >
bool smrat::cadcells::datastructures::SampledDerivation< Properties >::contains (
 const P & property) const [inline]
```

```
0.15.464.3.5 delin() template<typename Properties >
Delineation& smtrat::cadcells::datastructures::SampledDerivation< Properties >::delin ()
[inline]
```

```
0.15.464.3.6 delineate_cell() template<typename Properties >
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::delineate_cell ()
[inline]
Determines the cell w.r.t.
the delineation.
```

```
0.15.464.3.7 delineated() [1/2] template<typename Properties >
DelineatedDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation<
Properties >::delineated () [inline]
```

```
0.15.464.3.8 delineated() [2/2] template<typename Properties >
const DelineatedDerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation<
Properties >::delineated () const [inline]
```

```
0.15.464.3.9 insert() template<typename Properties >
template<typename P >
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::insert (
 P property) [inline]
```

```
0.15.464.3.10 level() template<typename Properties >
size_t smtrat::cadcells::datastructures::SampledDerivation< Properties >::level () const
[inline]
```

```
0.15.464.3.11 main_var() template<typename Properties >
carl::Variable smtrat::cadcells::datastructures::SampledDerivation< Properties >::main_var ()
const [inline]
```

```
0.15.464.3.12 main_var_sample() template<typename Properties >
const RAN& smtrat::cadcells::datastructures::SampledDerivation< Properties >::main_var_sample
() const [inline]
```

```
0.15.464.3.13 merge_with() template<typename Properties >
void smtrat::cadcells::datastructures::SampledDerivation< Properties >::merge_with (
 const SampledDerivation< Properties > & other) [inline]
```

```
0.15.464.3.14 polys() template<typename Properties >
PolyPool& smtrat::cadcells::datastructures::SampledDerivation< Properties >::polys () [inline]
```

```
0.15.464.3.15 proj() template<typename Properties >
Projections& smtrat::cadcells::datastructures::SampledDerivation< Properties >::proj ()
[inline]
```

```
0.15.464.3.16 properties() template<typename Properties >
template<typename P >
const PropertiesTSet<P>& smtrat::cadcells::datastructures::SampledDerivation< Properties >::properties () const [inline]
```

```
0.15.464.3.17 sample() template<typename Properties >
const Assignment& smtrat::cadcells::datastructures::SampledDerivation< Properties >::sample ()
const [inline]
```

```
0.15.464.3.18 underlying() template<typename Properties >
DerivationRef<Properties>& smtrat::cadcells::datastructures::SampledDerivation< Properties >::underlying () [inline]
```

```
0.15.464.3.19 underlying_sample() template<typename Properties >
const Assignment& smtrat::cadcells::datastructures::SampledDerivation< Properties >::underlying< _sample () const [inline]
```

## 0.15.465 smtrat::SampledDerivationRefCompare Struct Reference

```
#include <Helper.h>
```

### Public Member Functions

- template<typename T >  
constexpr bool operator() (const cadcells::datastructures::SampledDerivationRef< T > &a, const cadcells::datastructures::SampledDerivationRef< T > &b) const

### 0.15.465.1 Member Function Documentation

```
0.15.465.1.1 operator()() template<typename T >
constexpr bool smtrat::SampledDerivationRefCompare::operator() (
 const cadcells::datastructures::SampledDerivationRef< T > & a,
 const cadcells::datastructures::SampledDerivationRef< T > & b) const [inline],
[constexpr]
```

## 0.15.466 smtrat::cad::SampleIteratorQueue< Iterator, Comparator > Class Template Reference

```
#include <SampleIteratorQueue.h>
```

### Public Member Functions

- auto **begin** () const
- auto **end** () const
- template<typename InputIt >  
void **assign** (InputIt **begin**, InputIt **end**)
- void **clear** ()
- template<typename Filter >  
void **cleanup** (Filter &&f)
- bool **empty** () const
- Iterator **getNextSample** ()
- void **addNewSample** (Iterator it)

- template<typename InputIt >  
void **addNewSamples** (InputIt **begin**, InputIt **end**)
- Iterator **removeNextSample** ()
- bool **is\_consistent** () const

### 0.15.466.1 Member Function Documentation

**0.15.466.1.1 addNewSample()** template<typename Iterator , typename Comparator >  
void **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::addNewSample** (  
    Iterator **it** ) [inline]

**0.15.466.1.2 addNewSamples()** template<typename Iterator , typename Comparator >  
template<typename InputIt >  
void **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::addNewSamples** (  
    InputIt **begin**,  
    InputIt **end** ) [inline]

**0.15.466.1.3 assign()** template<typename Iterator , typename Comparator >  
template<typename InputIt >  
void **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::assign** (  
    InputIt **begin**,  
    InputIt **end** ) [inline]

**0.15.466.1.4 begin()** template<typename Iterator , typename Comparator >  
auto **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::begin** ( ) const [inline]

**0.15.466.1.5 cleanup()** template<typename Iterator , typename Comparator >  
template<typename Filter >  
void **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::cleanup** (  
    Filter && **f** ) [inline]

**0.15.466.1.6 clear()** template<typename Iterator , typename Comparator >  
void **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::clear** ( ) [inline]

**0.15.466.1.7 empty()** template<typename Iterator , typename Comparator >  
bool **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::empty** ( ) const [inline]

**0.15.466.1.8 end()** template<typename Iterator , typename Comparator >  
auto **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::end** ( ) const [inline]

**0.15.466.1.9 getNextSample()** template<typename Iterator , typename Comparator >  
Iterator **smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::getNextSample** ( ) [inline]

```
0.15.466.1.10 is_consistent() template<typename Iterator , typename Comparator >
bool smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::is_consistent () const [inline]
```

```
0.15.466.1.11 removeNextSample() template<typename Iterator , typename Comparator >
Iterator smtrat::cad::SampleIteratorQueue< Iterator, Comparator >::removeNextSample () [inline]
```

## 0.15.467 smtrat::sampling< S > Struct Template Reference

```
#include <Sampling.h>
```

### Static Public Member Functions

- template<typename T >  
static size\_t **sample\_outside** (cadcells::RAN &sample, const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare > &derivations)

### 0.15.467.1 Member Function Documentation

```
0.15.467.1.1 sample_outside() template<SamplingAlgorithm S>
template<typename T >
static size_t smtrat::sampling< S >::sample_outside (
 cadcells::RAN & sample,
 const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare > & derivations) [static]
```

## 0.15.468 smtrat::sampling< SamplingAlgorithm::LOWER\_UPPER\_BETWEEN\_SAMPLING > Struct Reference

```
#include <Sampling.h>
```

### Static Public Member Functions

- template<typename T >  
static size\_t **sample\_outside** (cadcells::RAN &sample, const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare > &derivationsSet)

### 0.15.468.1 Member Function Documentation

```
0.15.468.1.1 sample_outside() template<typename T >
static size_t smtrat::sampling< SamplingAlgorithm::LOWER_UPPER_BETWEEN_SAMPLING >::sample_←
outside (
 cadcells::RAN & sample,
 const std::set< cadcells::datastructures::SampledDerivationRef< T >, SampledDerivationRefCompare > & derivationsSet) [inline], [static]
```

## 0.15.469 smtrat::SATModule< Settings > Class Template Reference

Implements a module performing DPLL style SAT checking.

```
#include <SATModule.h>
```

## Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

## Public Member Functions

- `SATModule (const ModuleInput *_formula, Conditionals &_foundAnswer, Manager *const _manager=nullptr)`  
`Constructs a SATModule.`
- `~SATModule ()`  
`Destructs this SATModule.`
- `bool addCore (ModuleInput::const_iterator)`  
`The module has to take the given sub-formula of the received formula into account.`
- `Answer checkCore ()`  
`Checks the received formula for consistency.`
- `void removeCore (ModuleInput::const_iterator)`  
`Removes everything related to the given sub-formula of the received formula.`
- `void updateModel () const`  
`Updates the model, if the solver has detected the consistency of the received formula, beforehand.`
- `void updateAllModels ()`  
`Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.`
- `void updateInfeasibleSubset ()`  
`Updates the infeasible subset found by the SATModule, if the received formula is unsatisfiable.`
- `void cleanUpAfterOptimizing (const std::vector< Minisat::CRef > &_excludedAssignments)`
- `void removeUpperBoundOnMinimal ()`
- `void addConstraintToInform (const FormulaT &)`  
`Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.`
- `void addConstraintToInform_ (const FormulaT &_formula)`
- `void addBooleanAssignments (RationalAssignment &_rationalAssignment) const`  
`Adds the Boolean assignments to the given assignments, which were determined by the Minisat procedure.`
- `void print (std::ostream &_out=std::cout, const std::string &_init="") const`  
`Prints everything.`
- `void printCurrentAssignment (std::ostream &=std::cout, const std::string &=_init="") const`  
`Prints the current assignment of the SAT solver.`
- `void printConstraintLiteralMap (std::ostream &_out=std::cout, const std::string &_init="") const`  
`Prints the constraints to literal map.`
- `void printFormulaCNFInfosMap (std::ostream &_out=std::cout, const std::string &_init="") const`  
`Prints the formulas to clauses map.`
- `void printClauseInformation (std::ostream &_out=std::cout, const std::string &_init="") const`  
`Prints the clause information.`
- `void printBooleanVarMap (std::ostream &_out=std::cout, const std::string &_init="") const`  
`Prints map of the Boolean within the SAT solver to the given Booleans.`
- `void printBooleanConstraintMap (std::ostream &_out=std::cout, const std::string &_init="") const`  
`Prints the literal to constraint map.`
- `void printClause (Minisat::CRef _clause, bool _withAssignment=false, std::ostream &_out=std::cout, const std::string &_init="") const`  
`Prints the clause at the given reference.`
- `void printClause (const Minisat::vec< Minisat::Lit > &, bool _withAssignment=false, std::ostream &_out=std::cout, const std::string &_init="") const`  
`Prints the clause resulting from the given vector of literals.`
- `void printClauses (const Minisat::vec< Minisat::CRef > &_clauses, const std::string _name, std::ostream &_out=std::cout, const std::string &_init="", int=0, bool _withAssignment=false, bool _onlyNotSatisfied=false) const`

- Prints all given clauses.*
- void **printDecisions** (std::ostream &\_out=std::cout, const std::string &\_init="") const  
*Prints the decisions the SAT solver has made.*
  - void **printPropagatedLemmas** (std::ostream &\_out=std::cout, const std::string &\_init="") const  
*Prints the propagated lemmas for each variables which influence its value.*
  - void **printLiteralsActiveOccurrences** (std::ostream &\_out=std::cout, const std::string &\_init="") const  
*Prints the literals' active occurrences in all clauses.*
  - void **collectStats** ()  
*Collects the taken statistics.*
  - bool **inform** (const **FormulaT** &\_constraint)  
*Informs the module about the given constraint.*
  - void **deinform** (const **FormulaT** &\_constraint)  
*The inverse of informing about a constraint.*
  - virtual void **init** ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
  - bool **add** (**ModuleInput**::const\_iterator \_subformula)  
*The module has to take the given sub-formula of the received formula into account.*
  - virtual **Answer check** (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_← VARIABLE)  
*Checks the received formula for consistency.*
  - virtual void **remove** (**ModuleInput**::const\_iterator \_subformula)  
*Removes everything related to the given sub-formula of the received formula.*
  - virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
  - unsigned **currentlySatisfiedByBackend** (const **FormulaT** &\_formula) const
  - virtual unsigned **currentlySatisfied** (const **FormulaT** &) const
  - bool **receivedVariable** (carl::Variable::Arg \_var) const
  - **Answer solverState** () const
  - std::size\_t **id** () const
  - void **setId** (std::size\_t \_id)  
*Sets this modules unique ID to identify itself.*
  - **thread\_priority threadPriority** () const
  - void **setThreadPriority** (**thread\_priority** \_threadPriority)  
*Sets the priority of this module to get a thread for running its check procedure.*
  - const **ModuleInput** \* **pReceivedFormula** () const
  - const **ModuleInput** & **rReceivedFormula** () const
  - const **ModuleInput** \* **pPassedFormula** () const
  - const **ModuleInput** & **rPassedFormula** () const
  - const **Model** & **model** () const
  - const std::vector< **Model** > & **allModels** () const
  - const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
  - const std::vector< **Module** \* > & **usedBackends** () const
  - const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
  - const carl::FastSet< **FormulaT** > & **informedConstraints** () const
  - void **addLemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lt=Lemmatype::NORMAL, const **FormulaT** &\_preferredFormula=**FormulaT**(carl::FormulaType::TRUE))  
*Stores a lemma being a valid formula.*
  - bool **hasLemmas** ()  
*Checks whether this module or any of its backends provides any lemmas.*
  - void **clearLemmas** ()  
*Deletes all yet found lemmas.*
  - const std::vector< **Lemma** > & **lemmas** () const

- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- `void receivedFormulaChecked ()`  
*Notifies that the received formulas has been checked.*
- `const smrat::Conditionals & answerFound () const`
- `bool isPreprocessor () const`
- `virtual std::string moduleName () const`
- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`  
*Excludes all variables from the current model, which do not occur in the received formula.*
- `void updateLemmas ()`  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
*Collects the formulas in the given formula, which are part of the received formula.*
- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- `virtual std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- `void printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- `void printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- `void printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*
- `void printModel (std::ostream &_out=std::cout) const`  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- `void printAllModels (std::ostream &_out=std::cout)`  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- `static void freeSplittingVariable (const FormulaT &_splittingVariable)`

### Static Public Attributes

- `static size_t mNumOfBranchVarsToStore = 5`  
*The number of different variables to consider for a probable infinite loop of branchings.*
- `static std::vector< Branching > mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`  
*Stores the last branches in a cycle buffer.*
- `static size_t mFirstPosInLastBranches = 0`  
*The beginning of the cyclic buffer storing the last branches.*
- `static std::vector< FormulaT > mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)
 

*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)
 

*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const
 

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const
 

*Clears the assignment, if any was found.*
- void `clearModels` () const
 

*Clears all assignments, if any was found.*
- void `cleanModel` () const
 

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin` ()
- `ModuleInput::iterator passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
 

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const
 

*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const `Model` & `backendsModel` () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel` () const

- void `getBackendsAllModels () const`  
*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`  
*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`
- std::vector< `FormulaT` > `merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &)` const  
*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin (const std::vector< FormulaT > &origins) const`
- bool `probablyLooping (const typename Poly::PolyType &_branchingPolynomial, const Rational &_branchingValue) const`  
*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`  
*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt (const Poly &_polynomial, bool _integral, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &&_premise=std::vector< FormulaT >())`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, std::vector< FormulaT > &&_premise, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false)`
- bool `branchAt (carl::Variable::Arg _var, const Rational &_value, bool _leftCaseWeak=true, bool _preferLeftCase=true, bool _useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &_premise=std::vector< FormulaT >())`
- void `splitUnequalConstraint (const FormulaT &)`  
*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel () const`
- void `addInformationRelevantFormula (const FormulaT &formula)`  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas ()`  
*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel (LemmaLevel level)`  
*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint (const Model &_modelA, const Model &_modelB)`  
*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- `Manager *const mpManager`  
*A reference to the manager.*
- `Model mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< `Model` > `mAllModels`  
*Stores all satisfying assignments.*

- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- `Conditionals mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module *` > `mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module *` > `mAllBackends`  
*The backends of this module which have been used.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- carl::FastSet< `FormulaT` > `mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned `mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > `mVariableCounters`  
*Maps variables to the number of their occurrences.*

## Friends

- class `VarSchedulerBase`
- class `VarSchedulerMcsatBase`

### 0.15.469.1 Detailed Description

```
template<class Settings>
class smrat::SATModule< Settings >
```

Implements a module performing DPLL style SAT checking.

It is mainly based on [Minisat](#) 2.0 and extends it by enabling the SMT-RAT module interfaces and some optimizations.

### 0.15.469.2 Member Typedef Documentation

```
0.15.469.2.1 SettingsType template<class Settings >
typedef Settings smrat::SATModule< Settings >::SettingsType
```

### 0.15.469.3 Member Enumeration Documentation

#### 0.15.469.3.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.469.4 Constructor & Destructor Documentation

#### 0.15.469.4.1 SATModule() template<class Settings > smtrat::SATModule< Settings >::SATModule (

```
 const ModuleInput * _formula,
 Conditionals & _foundAnswer,
 Manager *const _manager = nullptr)
```

Constructs a **SATModule**.

Parameters

|                           |                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------|
| <code>_type</code>        | The type of this module being <b>SATModule</b> .                                            |
| <code>_formula</code>     | The formula passed to this module, called received formula.                                 |
| <code>_settings</code>    | [Not yet used.]                                                                             |
| <code>_foundAnswer</code> | Vector of Booleans: If any of them is true, we have to terminate a running check procedure. |
| <code>_manager</code>     | A reference to the manager of the solver using this module.                                 |

#### 0.15.469.4.2 ~SATModule() template<class Settings > smtrat::SATModule< Settings >::~SATModule ( )

Destructs this **SATModule**.

### 0.15.469.5 Member Function Documentation

#### 0.15.469.5.1 add() bool smtrat::Module::add (

```
 ModuleInput::const_iterator _subformula) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

#### 0.15.469.5.2 addBooleanAssignments() template<class Settings >

```
void smtrat::SATModule< Settings >::addBooleanAssignments (
 RationalAssignment & _rationalAssignment) const
Adds the Boolean assignments to the given assignments, which were determined by the Minisat procedure.
Note: Assignments in the given map are not overwritten.
```

#### Parameters

|                                  |                                                    |
|----------------------------------|----------------------------------------------------|
| <code>_rationalAssignment</code> | The assignments to add the Boolean assignments to. |
|----------------------------------|----------------------------------------------------|

**0.15.469.5.3 `addConstraintToInform()`** template<class Settings >  
void smtrat::SATModule< Settings >::addConstraintToInform (
 const FormulaT & \_constraint ) [inline], [virtual]  
Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

#### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.469.5.4 `addConstraintToInform_()`** template<class Settings >  
void smtrat::SATModule< Settings >::addConstraintToInform\_ (
 const FormulaT & \_formula ) [inline]

**0.15.469.5.5 `addCore()`** template<class Settings >  
bool smtrat::SATModule< Settings >::addCore (
 ModuleInput::const\_iterator ) [virtual]

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.469.5.6 `addInformationRelevantFormula()`** void smtrat::Module::addInformationRelevantFormula
(
 const FormulaT & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

```
0.15.469.5.7 addLemma() void smtrat::Module::addLemma (
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| _lemma            | The eduction/lemma to store.                                          |
| _lt               | The type of the lemma.                                                |
| _preferredFormula | Hint for the next decision, which formula should be assigned to true. |

```
0.15.469.5.8 addOrigin() void smtrat::Module::addOrigin (
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|          |                                                           |
|----------|-----------------------------------------------------------|
| _formula | The passed formula to set the origins for.                |
| _origin  | A set of formulas in the received formula of this module. |

```
0.15.469.5.9 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| _subformula | The sub-formula of the received formula to copy. |
|-------------|--------------------------------------------------|

```
0.15.469.5.10 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool>
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|          |                                           |
|----------|-------------------------------------------|
| _formula | The formula to add to the passed formula. |
|----------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.469.5.11 addSubformulaToPassedFormula()** [2/3] std::pair<ModuleInput::iterator,bool>  
smrat::Module::addSubformulaToPassedFormula (const FormulaT & \_formula, const FormulaT & \_origin) [inline], [protected], [inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                 |
| _origin  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.469.5.12 addSubformulaToPassedFormula()** [3/3] std::pair<ModuleInput::iterator,bool>

smrat::Module::addSubformulaToPassedFormula (const FormulaT & \_formula, const std::shared\_ptr<std::vector<FormulaT>> & \_origins) [inline], [protected], [inherited]

Adds the given formula to the passed formula.

#### Parameters

|          |                                                                                                                                         |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| _formula | The formula to add to the passed formula.                                                                                               |
| _origins | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.469.5.13 allModels()** const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]

#### Returns

All satisfying assignments, if existent.

**0.15.469.5.14 anAnswerFound()** bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

**0.15.469.5.15 answerFound()** const `smtrat::Conditionals`& `smtrat::Module::answerFound` ( ) const [inline], [inherited]

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.469.5.16 backendsModel()** const `Model` & `smtrat::Module::backendsModel` ( ) const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.469.5.17 branchAt() [1/4]** bool `smtrat::Module::branchAt` (  
`carl::Variable::Arg _var,`  
`const Rational & _value,`  
`bool _leftCaseWeak = true,`  
`bool _preferLeftCase = true,`  
`bool _useReceivedFormulaAsPremise = false,`  
`const std::vector< FormulaT > & _premise = std::vector<FormulaT>()` ) [inline], [protected], [inherited]

**0.15.469.5.18 branchAt() [2/4]** bool `smtrat::Module::branchAt` (  
`carl::Variable::Arg _var,`  
`const Rational & _value,`  
`std::vector< FormulaT > && _premise,`  
`bool _leftCaseWeak = true,`  
`bool _preferLeftCase = true,`  
`bool _useReceivedFormulaAsPremise = false`) [inline], [protected], [inherited]

**0.15.469.5.19 branchAt() [3/4]** bool `smtrat::Module::branchAt` (  
`const Poly & _polynomial,`  
`bool _integral,`  
`const Rational & _value,`  
`bool _leftCaseWeak = true,`  
`bool _preferLeftCase = true,`  
`bool _useReceivedFormulaAsPremise = false,`  
`const std::vector< FormulaT > & _premise = std::vector<FormulaT>()` ) [inline], [protected], [inherited]

**0.15.469.5.20 branchAt() [4/4]** bool `smtrat::Module::branchAt` (  
`const Poly & _polynomial,`  
`bool _integral,`  
`const Rational & _value,`  
`std::vector< FormulaT > && _premise,`  
`bool _leftCaseWeak = true,`  
`bool _preferLeftCase = true,`  
`bool _useReceivedFormulaAsPremise = false`) [protected], [inherited]

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding `SATModule`.

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                          |                             |
|--------------------------|-----------------------------|
| <code>_polynomial</code> | The variable to branch for. |
|--------------------------|-----------------------------|

## Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

### 0.15.469.5.21 `check()`

```
Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

## Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

## Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

### 0.15.469.5.22 `checkCore()`

```
template<class Settings >
Answer smtrat::SATModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

## Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

### 0.15.469.5.23 `checkInfSubsetForMinimality()`

```
void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

## Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.469.5.24 `checkModel()`** `unsigned smrat::Module::checkModel () const [protected], [inherited]`

## Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.469.5.25 `cleanModel()`** `void smrat::Module::cleanModel () const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.469.5.26 `cleanUpAfterOptimizing()`** `template<class Settings > void smrat::SATModule< Settings >::cleanUpAfterOptimizing ( const std::vector< Minisat::CRef > & _excludedAssignments )`

**0.15.469.5.27 `clearLemmas()`** `void smrat::Module::clearLemmas () [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.469.5.28 `clearModel()`** `void smrat::Module::clearModel () const [inline], [protected], [inherited]`  
Clears the assignment, if any was found.

**0.15.469.5.29 `clearModels()`** `void smrat::Module::clearModels () const [inline], [protected], [inherited]`  
Clears all assignments, if any was found.

**0.15.469.5.30 `clearPassedFormula()`** `void smrat::Module::clearPassedFormula () [protected], [inherited]`

**0.15.469.5.31 `collectOrigins() [1/2]`** `void smrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.469.5.32 `collectOrigins() [2/2]`** `void smrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.469.5.33 `collectStats()`** `template<class Settings >`  
`void smtrat::SATModule< Settings >::collectStats ( )`  
Collects the taken statistics.

**0.15.469.5.34 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations ( )`  
[inherited]

**0.15.469.5.35 `constraintsToInform()`** `const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]`

#### Returns

The constraints which the backends must be informed about.

**0.15.469.5.36 `currentlySatisfied()`** `virtual unsigned smtrat::Module::currentlySatisfied (`  
`const FormulaT & ) const [inline], [virtual], [inherited]`

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in `smtrat::LRAbstractModule< Settings >`, `smtrat::LRAbstractModule< LRASettingsICP >`, and `smtrat::LRAbstractModule< LRASettings >`.

**0.15.469.5.37 `currentlySatisfiedByBackend()`** `unsigned smtrat::Module::currentlySatisfiedByBackend (`  
`const FormulaT & _formula ) const [inherited]`

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.469.5.38 `deinform()`** `void smtrat::Module::deinform (`  
`const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.469.5.39 `deinformCore()`** `virtual void smtrat::Module::deinformCore (`  
 `const FormulaT & _constraint )` [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.469.5.40 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (`  
 `const std::vector< FormulaT > & origins )` const [protected], [inherited]

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.469.5.41 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`  
 `ModuleInput::iterator _subformula,`  
 `bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
 Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.469.5.42 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( )` const [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.469.5.43 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`  
 `const std::vector< FormulaT > & _origins )` const [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

**0.15.469.5.44 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.469.5.45 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.469.5.46 `freeSplittingVariable()`** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable) [inline], [static], [inherited]`

**0.15.469.5.47 `generateTrivialInfeasibleSubset()`** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.469.5.48 `getBackendsAllModels()`** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.469.5.49 `getBackendsModel()`** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.469.5.50 `getInfeasibleSubsets() [1/2]`** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.469.5.51 `getInfeasibleSubsets() [2/2]`** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.469.5.52 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.469.5.53 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.469.5.54 `getOrigins() [1/3]`** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.469.5.55 `getOrigins() [2/3]`** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.469.5.56 `getOrigins() [3/3]`** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.469.5.57 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT >` `smtrat::Module::getReceivedFormulaSimplified( )` [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.469.5.58 `hasLemmas()`** `bool smtrat::Module::hasLemmas( )` [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.469.5.59 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const` [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.469.5.60 `id()`** `std::size_t smtrat::Module::id( ) const` [inline], [inherited]

**Returns**

A unique ID to identify this module instance.

**0.15.469.5.61 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const` [inline], [inherited]

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.469.5.62 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint )` [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before `assertSubformula` is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.469.5.63 informBackends()** void smrat::Module::informBackends ( const FormulaT & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**0.15.469.5.64 informCore()** virtual bool smrat::Module::informCore (

const FormulaT & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.469.5.65 informedConstraints()** const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.469.5.66 init()** void smrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

**0.15.469.5.67 is\_minimizing()** bool smrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.469.5.68 isLemmaLevel()** `bool smtrat::Module::isLemmaLevel (``LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.469.5.69 isPreprocessor()** `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.469.5.70 lemmas()** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.469.5.71 merge()** `std::vector< FormulaT > smtrat::Module::merge (``const std::vector< FormulaT > & _vecSetA,``const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.469.5.72 model()** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.469.5.73 modelsDisjoint()** `bool smtrat::Module::modelsDisjoint (``const Model & _modelA,``const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.469.5.74 `moduleName()`** `virtual std::string smtrat::Module::moduleName( ) const [inline], [virtual], [inherited]`

**Returns**

The name of the given module type as name.

Reimplemented in `smtrat::VSModule< Settings >`, `smtrat::UnionFindModule< Settings >`, `smtrat::UFCegarModule< Settings >`, `smtrat::STropModule< Settings >`, `smtrat::SplitSOSModule< Settings >`, `smtrat::PBPPModule< Settings >`, `smtrat::PBGaussModule< Settings >`, `smtrat::NRAILModule< Settings >`, `smtrat::NewCADModule< Settings >`, `smtrat::LVEModule< Settings >`, `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, `smtrat::LRAModule< LRASettings1 >`, `smtrat::IntEqModule< Settings >`, `smtrat::IntBlastModule< Settings >`, `smtrat::IncWidthModule< Settings >`, `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, `smtrat::ICPModule< ICPSettings1 >`, `smtrat::GBModule< Settings >`, `smtrat::FouMoModule< Settings >`, `smtrat::EQPreprocessingModule< Settings >`, `smtrat::EQModule< Settings >`, `smtrat::CurryModule< Settings >`, `smtrat::CubeLIAModule< Settings >`, `smtrat::CSplitModule< Settings >`, `smtrat::BVMModule< Settings >`, and `smtrat::BEModule< Settings >`.

**0.15.469.5.75 `objective()`** `carl::Variable smtrat::Module::objective( ) const [inline], [inherited]`

**0.15.469.5.76 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.469.5.77 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.469.5.78 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.469.5.79 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.469.5.80 pReceivedFormula()** const `ModuleInput*` smtrat::Module::pReceivedFormula ( ) const  
[inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.469.5.81 print() [1/2]** void smtrat::Module::print (const std::string & *\_initiation* = "\*\*\*") const [inherited]  
Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.469.5.82 print() [2/2]** template<class Settings> void smtrat::SATModule< Settings >::print (std::ostream & *\_out* = std::cout, const std::string & *\_init* = "") const

Prints everything.

**Parameters**

|              |                                                       |
|--------------|-------------------------------------------------------|
| <i>_out</i>  | The output stream where the answer should be printed. |
| <i>_init</i> | The line initiation.                                  |

**0.15.469.5.83 printAllModels()** void smtrat::Module::printAllModels (std::ostream & *\_out* = std::cout) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.469.5.84 printBooleanConstraintMap()** template<class Settings> void smtrat::SATModule< Settings >::printBooleanConstraintMap (std::ostream & *\_out* = std::cout, const std::string & *\_init* = "") const

Prints the literal to constraint map.

**Parameters**

|              |                                                       |
|--------------|-------------------------------------------------------|
| <i>_out</i>  | The output stream where the answer should be printed. |
| <i>_init</i> | The line initiation.                                  |

**0.15.469.5.85 printBooleanVarMap()** template<class Settings>

```
void smtrat::SATModule< Settings >::printBooleanVarMap (
 std::ostream & _out = std::cout,
 const std::string & _init = "") const
```

Prints map of the Boolean within the SAT solver to the given Booleans.

#### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <code>_out</code>  | The output stream where the answer should be printed. |
| <code>_init</code> | The line initiation.                                  |

#### 0.15.469.5.86 `printClause()` [1/2] template<class Settings >

```
void smtrat::SATModule< Settings >::printClause (
 const Minisat::vec< Minisat::Lit > & ,
 bool _withAssignment = false,
 std::ostream & _out = std::cout,
 const std::string & _init = "") const
```

Prints the clause resulting from the given vector of literals.

#### Parameters

|                              |                                                                        |
|------------------------------|------------------------------------------------------------------------|
| <code>_clause</code>         | The reference of the clause.                                           |
| <code>_withAssignment</code> | A flag indicating if true, that the assignments should be printed too. |
| <code>_out</code>            | The output stream where the answer should be printed.                  |
| <code>_init</code>           | The prefix of each printed line.                                       |

#### 0.15.469.5.87 `printClause()` [2/2] template<class Settings >

```
void smtrat::SATModule< Settings >::printClause (
 Minisat::CRef _clause,
 bool _withAssignment = false,
 std::ostream & _out = std::cout,
 const std::string & _init = "") const
```

Prints the clause at the given reference.

#### Parameters

|                              |                                                                        |
|------------------------------|------------------------------------------------------------------------|
| <code>_clause</code>         | The reference of the clause.                                           |
| <code>_withAssignment</code> | A flag indicating if true, that the assignments should be printed too. |
| <code>_out</code>            | The output stream where the answer should be printed.                  |
| <code>_init</code>           | The prefix of each printed line.                                       |

#### 0.15.469.5.88 `printClauseInformation()` template<class Settings >

```
void smtrat::SATModule< Settings >::printClauseInformation (
 std::ostream & _out = std::cout,
 const std::string & _init = "") const
```

Prints the clause information.

#### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <code>_out</code>  | The output stream where the answer should be printed. |
| <code>_init</code> | The line initiation.                                  |

```
0.15.469.5.89 printClauses() template<class Settings >
void smtrat::SATModule< Settings >::printClauses (
 const Minisat::vec< Minisat::CRef > & _clauses,
 const std::string & _name,
 std::ostream & _out = std::cout,
 const std::string & _init = "",
 int = 0,
 bool _withAssignment = false,
 bool _onlyNotSatisfied = false) const
```

Prints all given clauses.

#### Parameters

|                              |                                                                               |
|------------------------------|-------------------------------------------------------------------------------|
| <code>_clauses</code>        | The clauses to print.                                                         |
| <code>_name</code>           | The name of the clauses to print. (e.g. learnts, clauses, ..)                 |
| <code>_out</code>            | The output stream where the answer should be printed.                         |
| <code>_init</code>           | The prefix of each printed line.                                              |
| <code>_from</code>           | The position of the first clause to print within the given vector of clauses. |
| <code>_withAssignment</code> | A flag indicating if true, that the assignments should be printed too.        |

```
0.15.469.5.90 printConstraintLiteralMap() template<class Settings >
void smtrat::SATModule< Settings >::printConstraintLiteralMap (
 std::ostream & _out = std::cout,
 const std::string & _init = "") const
```

Prints the constraints to literal map.

#### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <code>_out</code>  | The output stream where the answer should be printed. |
| <code>_init</code> | The line initiation.                                  |

```
0.15.469.5.91 printCurrentAssignment() template<class Settings >
void smtrat::SATModule< Settings >::printCurrentAssignment (
 std::ostream & = std::cout,
 const std::string & = "") const
```

Prints the current assignment of the SAT solver.

#### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <code>_out</code>  | The output stream where the answer should be printed. |
| <code>_init</code> | The line initiation.                                  |

```
0.15.469.5.92 printDecisions() template<class Settings >
void smtrat::SATModule< Settings >::printDecisions (
 std::ostream & _out = std::cout,
 const std::string & _init = "") const
```

Prints the decisions the SAT solver has made.

**Parameters**

|              |                                                       |
|--------------|-------------------------------------------------------|
| <i>_out</i>  | The output stream where the answer should be printed. |
| <i>_init</i> | The line initiation.                                  |

**0.15.469.5.93 printFormulaCNFInfosMap()** template<class Settings >  
void smtrat::SATModule< Settings >::printFormulaCNFInfosMap ( std::ostream & *\_out* = std::cout,  
const std::string & *\_init* = "" ) const  
Prints the formulas to clauses map.

**Parameters**

|              |                                                       |
|--------------|-------------------------------------------------------|
| <i>_out</i>  | The output stream where the answer should be printed. |
| <i>_init</i> | The line initiation.                                  |

**0.15.469.5.94 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.469.5.95 printLiteralsActiveOccurrences()** template<class Settings >  
void smtrat::SATModule< Settings >::printLiteralsActiveOccurrences ( std::ostream & *\_out* = std::cout,  
const std::string & *\_init* = "" ) const  
Prints the literals' active occurrences in all clauses.

**Parameters**

|              |                                                       |
|--------------|-------------------------------------------------------|
| <i>_out</i>  | The output stream where the answer should be printed. |
| <i>_init</i> | The line initiation.                                  |

**0.15.469.5.96 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.469.5.97 printPassedFormula()** void smtrat::Module::printPassedFormula (

---

```
 const std::string & _initiation = "***") const [inherited]
Prints the vector of passed formula.
```

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.469.5.98 printPropagatedLemmas()** template<class Settings >  
void smrat::SATModule< Settings >::printPropagatedLemmas (

```
 std::ostream & _out = std::cout,
 const std::string & _init = "") const
```

Prints the propagated lemmas for each variables which influence its value.

**Parameters**

|              |                                                       |
|--------------|-------------------------------------------------------|
| <i>_out</i>  | The output stream where the answer should be printed. |
| <i>_init</i> | The line initiation.                                  |

**0.15.469.5.99 printReceivedFormula()** void smrat::Module::printReceivedFormula (

```
 const std::string & _initiation = "***") const [inherited]
```

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.469.5.100 probablyLooping()** bool smrat::Module::probablyLooping (

```
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

**Parameters**

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

**Returns**

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.469.5.101 receivedFormulaChecked()** void smrat::Module::receivedFormulaChecked ( ) [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.469.5.102 receivedFormulasAsInfeasibleSubset()** void smrat::Module::receivedFormulasAsInfeasibleSubset (

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

**0.15.469.5.103 `receivedVariable()`** `bool smtrat::Module::receivedVariable ( carl::Variable::Arg _var ) const [inline], [inherited]`

**0.15.469.5.104 `remove()`** `void smtrat::Module::remove ( ModuleInput::const_iterator _subformula ) [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.469.5.105 `removeCore()`** `template<class Settings > void smtrat::SATModule< Settings >::removeCore ( ModuleInput::const_iterator ) [virtual]`

Removes everything related to the given sub-formula of the received formula.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.469.5.106 `removeOrigin()`** `std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.469.5.107 `removeOrigins()`** `std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

**0.15.469.5.108 `removeUpperBoundOnMinimal()`** `template<class Settings > void smtrat::SATModule< Settings >::removeUpperBoundOnMinimal ( )`

**0.15.469.5.109 `rPassedFormula()`** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.469.5.110 rReceivedFormula()** const `ModuleInput`& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]

**Returns**

A reference to the formula passed to this module.

**0.15.469.5.111 runBackends() [1/2]** virtual `Answer` smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited]  
Reimplemented in [smtrat::PModule](#).

**0.15.469.5.112 runBackends() [2/2]** `Answer` smtrat::Module::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.469.5.113 setId()** void smtrat::Module::setId ( std::size\_t `_id` ) [inline], [inherited]

Sets this modules unique ID to identify itself.

**Parameters**

|                                                                       |                                    |
|-----------------------------------------------------------------------|------------------------------------|
| $\leftrightarrow$<br>$\underline{\leftrightarrow}$<br><code>id</code> | The id to set this module's id to. |
|-----------------------------------------------------------------------|------------------------------------|

**0.15.469.5.114 setThreadPriority()** void smtrat::Module::setThreadPriority ( `thread_priority` `_threadPriority` ) [inline], [inherited]

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.469.5.115 solverState()** `Answer` smtrat::Module::solverState ( ) const [inline], [inherited]

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.469.5.116 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.469.5.117 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.469.5.118 `updateAllModels()`** `template<class Settings >`

`void smtrat::SATModule< Settings >::updateAllModels ( ) [virtual]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from `smtrat::Module`.

**0.15.469.5.119 `updateInfeasibleSubset()`** `template<class Settings >`

`void smtrat::SATModule< Settings >::updateInfeasibleSubset ( )`

Updates the infeasible subset found by the `SATModule`, if the received formula is unsatisfiable.

**0.15.469.5.120 `updateLemmas()`** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.469.5.121 `updateModel()`** `template<class Settings >`

`void smtrat::SATModule< Settings >::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from `smtrat::Module`.

**0.15.469.5.122 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.469.6 Friends And Related Function Documentation

**0.15.469.6.1 VarSchedulerBase** template<class Settings >  
friend class [VarSchedulerBase](#) [friend]

**0.15.469.6.2 VarSchedulerMcsatBase** template<class Settings >  
friend class [VarSchedulerMcsatBase](#) [friend]

## 0.15.469.7 Field Documentation

**0.15.469.7.1 mAllBackends** std::vector<[Module](#)\*> smrat::Module::mAllBackends [protected],  
[inherited]

The backends of this module which have been used.

**0.15.469.7.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected],  
[inherited]

Stores all satisfying assignments.

**0.15.469.7.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer  
[protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.469.7.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform  
[protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.469.7.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.469.7.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.469.7.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smrat::Module::mFirstSubformulaTo→  
Pass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.469.7.8 mFirstUncheckedReceivedSubformula** [ModuleInput](#)::const\_iterator smrat::Module→  
::mFirstUncheckedReceivedSubformula [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.469.7.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.469.7.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.469.7.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets`  
[protected], [inherited]  
Stores the infeasible subsets.

**0.15.469.7.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints`  
[protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.469.7.13 mLastBranches** `std::vector< Branching >` `smtrat::Module::mLastBranches` = `std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static],  
[inherited]  
Stores the last branches in a cycle buffer.

**0.15.469.7.14 mModel** `Model` `smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.469.7.15 mModelComputed** `bool` `smtrat::Module::mModelComputed` [mutable], [protected],  
[inherited]  
True, if the model has already been computed.

**0.15.469.7.16 mNumOfBranchVarsToStore** `std::size_t` `smtrat::Module::mNumOfBranchVarsToStore` =  
5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.469.7.17 mObjectiveVariable** `carl::Variable` `smtrat::Module::mObjectiveVariable` [protected],  
[inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.469.7.18 mOldSplittingVariables** `std::vector< FormulaT >` `smtrat::Module::mOldSplittingVariables`  
[static], [inherited]  
Reusable splitting variables.

**0.15.469.7.19 mpManager** `Manager*` `const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.469.7.20 mSmallerMusesCheckCounter** `unsigned` `smtrat::Module::mSmallerMusesCheckCounter`  
[mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.469.7.21 mSolverState** std::atomic<[Answer](#)> smrat::Module::mSolverState [protected],  
[inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.469.7.22 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smrat::Module::mTheory←  
Propagations [protected], [inherited]

**0.15.469.7.23 mUsedBackends** std::vector<[Module\\*](#)> smrat::Module::mUsedBackends [protected],  
[inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.469.7.24 mVariableCounters** std::vector<std::size\_t> smrat::Module::mVariableCounters  
[protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.470 smrat::parser::ScriptParser< Callee > Struct Template Reference

#include <Script.h>

### Public Member Functions

- [ScriptParser \(InstructionHandler &h, Theories &theories, Callee &callee\)](#)
- void [startFunctionDefinition \(\)](#)

### Data Fields

- [InstructionHandler & handler](#)
- [Callee & callee](#)
- [ParserState state](#)
- [Theories & theories](#)
- [LogicParser logic](#)
- [AttributeParser attribute](#)
- [KeywordParser keyword](#)
- [NumeralParser numeral](#)
- [DecimalParser decimal](#)
- [QEParser qeQuery](#)
- [SortParser sort](#)
- [SortedVariableParser sortedvariable](#)
- [StringParser string](#)
- [SymbolParser symbol](#)
- [TermParser term](#)
- [qi::rule< Iterator, types::VariableType\(\), Skipper > functionDefinitionArg](#)
- [qi::rule< Iterator, Skipper > functionDefinition](#)
- [qi::rule< Iterator, Skipper > command](#)
- [qi::rule< Iterator, Skipper > main](#)
- [px::function< ErrorHandler > errorHandler](#)

### 0.15.470.1 Constructor & Destructor Documentation

```
0.15.470.1.1 ScriptParser() template<typename Callee >
smtrat::parser::ScriptParser< Callee >::ScriptParser (
 InstructionHandler & h,
 Theories & theories,
 Callee & callee) [inline]
```

## 0.15.470.2 Member Function Documentation

```
0.15.470.2.1 startFunctionDefinition() template<typename Callee >
void smtrat::parser::ScriptParser< Callee >::startFunctionDefinition () [inline]
```

## 0.15.470.3 Field Documentation

```
0.15.470.3.1 attribute template<typename Callee >
AttributeParser smtrat::parser::ScriptParser< Callee >::attribute
```

```
0.15.470.3.2 callee template<typename Callee >
Callee& smtrat::parser::ScriptParser< Callee >::callee
```

```
0.15.470.3.3 command template<typename Callee >
qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::command
```

```
0.15.470.3.4 decimal template<typename Callee >
DecimalParser smtrat::parser::ScriptParser< Callee >::decimal
```

```
0.15.470.3.5 errorHandler template<typename Callee >
px::function<ErrorHandler> smtrat::parser::ScriptParser< Callee >::errorHandler
```

```
0.15.470.3.6 functionDefinition template<typename Callee >
qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::functionDefinition
```

```
0.15.470.3.7 functionDefinitionArg template<typename Callee >
qi::rule<Iterator, types::VariableType(), Skipper> smtrat::parser::ScriptParser< Callee >::functionDefinitionArg
```

```
0.15.470.3.8 handler template<typename Callee >
InstructionHandler& smtrat::parser::ScriptParser< Callee >::handler
```

```
0.15.470.3.9 keyword template<typename Callee >
KeywordParser smtrat::parser::ScriptParser< Callee >::keyword
```

**0.15.470.3.10 logic** template<typename Callee >  
LogicParser smtrat::parser::ScriptParser< Callee >::logic

**0.15.470.3.11 main** template<typename Callee >  
qi::rule<Iterator, Skipper> smtrat::parser::ScriptParser< Callee >::main

**0.15.470.3.12 numeral** template<typename Callee >  
NumeralParser smtrat::parser::ScriptParser< Callee >::numeral

**0.15.470.3.13 qeQuery** template<typename Callee >  
QEParser smtrat::parser::ScriptParser< Callee >::qeQuery

**0.15.470.3.14 sort** template<typename Callee >  
SortParser smtrat::parser::ScriptParser< Callee >::sort

**0.15.470.3.15 sortedvariable** template<typename Callee >  
SortedVariableParser smtrat::parser::ScriptParser< Callee >::sortedvariable

**0.15.470.3.16 state** template<typename Callee >  
ParserState smtrat::parser::ScriptParser< Callee >::state

**0.15.470.3.17 string** template<typename Callee >  
StringParser smtrat::parser::ScriptParser< Callee >::string

**0.15.470.3.18 symbol** template<typename Callee >  
SymbolParser smtrat::parser::ScriptParser< Callee >::symbol

**0.15.470.3.19 term** template<typename Callee >  
TermParser smtrat::parser::ScriptParser< Callee >::term

**0.15.470.3.20 theories** template<typename Callee >  
Theories& smtrat::parser::ScriptParser< Callee >::theories

## 0.15.471 smtrat::parser::ParserState::ScriptScope Struct Reference

#include <ParserState.h>

### Public Member Functions

- `ScriptScope (const ParserState &state)`
- `void discharge (ParserState &state)`

#### 0.15.471.1 Constructor & Destructor Documentation

---

**0.15.471.1.1 ScriptScope()** `smtrat::parser::ParserState::ScriptScope::ScriptScope ( const ParserState & state ) [inline]`

## 0.15.471.2 Member Function Documentation

**0.15.471.2.1 discharge()** `void smtrat::parser::ParserState::ScriptScope::discharge ( ParserState & state ) [inline]`

## 0.15.472 smtrat::mcsat::onecellcad::Section Struct Reference

Represent a cell's (closed-interval-boundary) component along th k-th axis.

```
#include <utils.h>
```

### Data Fields

- `MultivariateRootT boundFunction`
- `RAN isolatedRoot`

*For performance we cache the isolated root from 'boundFunction' that lies closest along this section's level to the main point.*

### 0.15.472.1 Detailed Description

Represent a cell's (closed-interval-boundary) component along th k-th axis.

A section is a function  $f: \text{algReal}^{\{k-1\}} \rightarrow \text{algReal}$ ; from a multi-dimensional input point of level  $k-1$  (whose components are algebraic reals) to an algebraic real. We use a root-expression with an irreducible, multivariate polynomial of level  $k$ .

### 0.15.472.2 Field Documentation

**0.15.472.2.1 boundFunction** `MultivariateRootT smtrat::mcsat::onecellcad::Section::boundFunction`

**0.15.472.2.2 isolatedRoot** `RAN smtrat::mcsat::onecellcad::Section::isolatedRoot`

For performance we cache the isolated root from 'boundFunction' that lies closest along this section's level to the main point.

## 0.15.473 smtrat::onecellcad::recursive::Section Struct Reference

Represent a cell's closed-interval-boundary object along one single axis by an irreducible, multivariate polynomial of level  $k$ .

```
#include <OpenCAD.h>
```

### Data Fields

- `MultiPoly poly`
- `RAN cachedPoint`

*A single, special bound after having plugged a specific point of level  $k-1$  can be cached for performance (needed for [1]).*

### 0.15.473.1 Detailed Description

Represent a cell's closed-interval-boundary object along one single axis by an irreducible, multivariate polynomial of level k.

A section is an algebraic/"moving" boundary, because it's basically a function f: algReal<sup>{k-1}</sup> -> algReal; from a multi-dimensional input point of level k-1 (whose components are algebraic reals) to an algebraic real (the bound along k-th axis that changes depending on the input point).

### 0.15.473.2 Field Documentation

#### 0.15.473.2.1 **cachedPoint** RAN smtrat::oncellcad::recursive::Section::cachedPoint

A single, special bound after having plugged a specific point of level k-1 can be cached for performance (needed for [1]).

#### 0.15.473.2.2 **poly** MultiPoly smtrat::oncellcad::recursive::Section::poly

### 0.15.474 smtrat::mcsat::oncellcad::levelwise::SectionHeuristic1 Struct Reference

```
#include <Explanation.h>
```

#### Static Public Attributes

- static constexpr int **sectionHeuristic** = 1

### 0.15.474.1 Field Documentation

#### 0.15.474.1.1 **sectionHeuristic** constexpr int smtrat::mcsat::oncellcad::levelwise::Section<→ Heuristic1::sectionHeuristic = 1 [static], [constexpr]

### 0.15.475 smtrat::mcsat::oncellcad::levelwise::SectionHeuristic2 Struct Reference

```
#include <Explanation.h>
```

#### Static Public Attributes

- static constexpr int **sectionHeuristic** = 2

### 0.15.475.1 Field Documentation

#### 0.15.475.1.1 **sectionHeuristic** constexpr int smtrat::mcsat::oncellcad::levelwise::Section<→ Heuristic2::sectionHeuristic = 2 [static], [constexpr]

### 0.15.476 smtrat::mcsat::oncellcad::levelwise::SectionHeuristic3 Struct Reference

```
#include <Explanation.h>
```

#### Static Public Attributes

- static constexpr int **sectionHeuristic** = 3

### 0.15.476.1 Field Documentation

**0.15.476.1.1 sectionHeuristic** `constexpr int smtrat::mcsat::onecellcad::levelwise::Section::Heuristic3::sectionHeuristic = 3 [static], [constexpr]`

## 0.15.477 smtrat::mcsat::onecellcad::Sector Struct Reference

Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.

```
#include <utils.h>
```

### Data Fields

- `std::optional<Section> lowBound`  
*A std::nullopt lowBound represents negative infinity.*
- `std::optional<Section> highBound`  
*A std::nullopt highBound represents infinity.*

### 0.15.477.1 Detailed Description

Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.

A sector is an algebraic/"moving" boundary, because it's basically a function  $f: \text{algReal}^{k-1} \rightarrow (\text{algReal}, \text{algReal})$ ; from a multi-dimensional input point of level k-1 (whose components are algebraic reals) to a pair of algebraic reals (the lower and upper bound for an open interval along k-th axis that changes depending on the input point). Note that if 'lowBound' or 'highBound' is not defined, then this represents negative and positive infinity, respectively.

### 0.15.477.2 Field Documentation

**0.15.477.2.1 highBound** `std::optional<Section> smtrat::mcsat::onecellcad::Sector::highBound`  
A std::nullopt highBound represents infinity.

**0.15.477.2.2 lowBound** `std::optional<Section> smtrat::mcsat::onecellcad::Sector::lowBound`  
A std::nullopt lowBound represents negative infinity.

## 0.15.478 smtrat::onecellcad::recursive::Sector Struct Reference

Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.

```
#include <OpenCAD.h>
```

### Public Member Functions

- `bool isLowBoundNegInfty () const`
- `bool isHighBoundInfty () const`

### Data Fields

- `std::optional<Section> lowBound`
- `std::optional<Section> highBound`

### 0.15.478.1 Detailed Description

Represent a cell's open-interval boundary object along one single axis by two irreducible, multivariate polynomials of level k.

A sector is an algebraic/"moving" boundary, because it's basically a function f:  $\text{algReal}^{\wedge\{k-1\}} \rightarrow (\text{algReal}, \text{algReal})$ ; from a multi-dimensional input point of level k-1 (whose components are algebraic reals) to a pair of algebraic reals (the lower and upper bound for an open interval along k-th axis that changes depending on the input point). Note that if 'lowBound' or 'highBound' is not defined, then this represents negative and positive infinity, respectively.

### 0.15.478.2 Member Function Documentation

**0.15.478.2.1 `isHighBoundInfty()`** `bool smtrat::oncellcad::recursive::Sector::isHighBoundInfty () const [inline]`

**0.15.478.2.2 `isLowBoundNegInfty()`** `bool smtrat::oncellcad::recursive::Sector::isLowBoundNegInfty () const [inline]`

### 0.15.478.3 Field Documentation

**0.15.478.3.1 `highBound`** `std::optional<Section> smtrat::oncellcad::recursive::Sector::highBound`

**0.15.478.3.2 `lowBound`** `std::optional<Section> smtrat::oncellcad::recursive::Sector::lowBound`

## 0.15.479 `smtrat::mcsat::oncellcad::levelwise::SectorHeuristic1` Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr int `sectorHeuristic` = 1

### 0.15.479.1 Field Documentation

**0.15.479.1.1 `sectorHeuristic`** `constexpr int smtrat::mcsat::oncellcad::levelwise::SectorHeuristic1::sectorHeuristic = 1 [static], [constexpr]`

## 0.15.480 `smtrat::mcsat::oncellcad::levelwise::SectorHeuristic2` Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr int `sectorHeuristic` = 2

### 0.15.480.1 Field Documentation

```
0.15.480.1.1 sectorHeuristic constexpr int smtrat::mcsat::onecellcad::levelwise::SectorHeuristic2<→
::sectorHeuristic = 2 [static], [constexpr]
```

## 0.15.481 smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3 Struct Reference

```
#include <Explanation.h>
```

### Static Public Attributes

- static constexpr int **sectorHeuristic** = 3

### 0.15.481.1 Field Documentation

```
0.15.481.1.1 sectorHeuristic constexpr int smtrat::mcsat::onecellcad::levelwise::SectorHeuristic3<→
::sectorHeuristic = 3 [static], [constexpr]
```

## 0.15.482 smtrat::subtropical::Separator Struct Reference

Represents the class of all original constraints with the same left hand side after a normalization.

```
#include <Subtropical.h>
```

### Public Member Functions

- **Separator** (const **Poly** &normalization)

### Data Fields

- const carl::Variable **bias**  
*Bias variable of the separating hyperplane.*
- const std::vector<**Vertex**> **vertices**  
*Vertices for all terms of the normalized left hand side.*

### 0.15.482.1 Detailed Description

Represents the class of all original constraints with the same left hand side after a normalization.

Here, the set of all received relations of constraints with the same left hand side is stored. At any one time only one relation can be active and used for linearization.

### 0.15.482.2 Constructor & Destructor Documentation

```
0.15.482.2.1 Separator() smtrat::subtropical::Separator::Separator (
 const Poly & normalization) [inline]
```

### 0.15.482.3 Field Documentation

```
0.15.482.3.1 bias const carl::Variable smtrat::subtropical::Separator::bias
Bias variable of the separating hyperplane.
```

```
0.15.482.3.2 vertices const std::vector<Vertex> smtrat::subtropical::Separator::vertices
Vertices for all terms of the normalized left hand side.
```

### 0.15.483 `smtrat::mcsat::SequentialAssignment`< Backends > Struct Template Reference

```
#include <SequentialAssignment.h>
```

### 0.15.484 `smtrat::mcsat::SequentialExplanation`< Backends > Struct Template Reference

```
#include <SequentialExplanation.h>
```

### 0.15.485 `benchmax::settings::Settings` Struct Reference

Generic class to manage runtime settings.

```
#include <Settings.h>
```

#### 0.15.485.1 Detailed Description

Generic class to manage runtime settings.

Essentially stores all (named) settings in a map<string,any> and retrieves settings classes when requested.

### 0.15.486 `delta::Settings` Class Reference

This class loads and checks the command line options with help of `boost::program_options`.

```
#include <Settings.h>
```

#### Public Member Functions

- `Settings` (const std::string &executable)  
*Constructor.*
- bool `load` (int argc, char \*argv[ ])  
*Load options from command line.*
- template<typename T >  
const T & `as` (const std::string &s) const  
*Get option with given name as a given type.*
- template<typename T >  
void `set` (const std::string &s, const T &t)  
*Set option with given name as a given type.*
- bool `has` (const std::string &s) const  
*Checks if there is an option with this name.*
- bool `isDefault` (const std::string &s) const  
*Check if the option with the given name was set due to its default value.*

#### Friends

- std::ostream & `operator<<` (std::ostream &os, const `Settings` &s)  
*Streaming operator.*

#### 0.15.486.1 Detailed Description

This class loads and checks the command line options with help of `boost::program_options`.

#### 0.15.486.2 Constructor & Destructor Documentation

**0.15.486.2.1 Settings()** `delta::Settings::Settings (`  
`const std::string & executable ) [inline]`

Constructor.

**Parameters**

|                         |                      |
|-------------------------|----------------------|
| <code>executable</code> | Name of this binary. |
|-------------------------|----------------------|

**0.15.486.3 Member Function Documentation**

**0.15.486.3.1 `as()`** `template<typename T >`  
`const T& delta::Settings::as (`  
    `const std::string & s ) const [inline]`

Get option with given name as a given type.

If there is no such option or it has a different type, exceptions may be thrown.

**Parameters**

|                |                     |
|----------------|---------------------|
| <code>s</code> | Name of the option. |
|----------------|---------------------|

**Returns**

Value of the option as type T.

**0.15.486.3.2 `has()`** `bool delta::Settings::has (`  
    `const std::string & s ) const [inline]`

Checks if there is an option with this name.

**Parameters**

|                |                     |
|----------------|---------------------|
| <code>s</code> | Name of the option. |
|----------------|---------------------|

**Returns**

If option was set.

**0.15.486.3.3 `isDefault()`** `bool delta::Settings::isDefault (`  
    `const std::string & s ) const [inline]`

Check if the option with the given name was set due to its default value.

If there is no such option exceptions may be thrown.

**Parameters**

|                |                     |
|----------------|---------------------|
| <code>s</code> | Name of the option. |
|----------------|---------------------|

**Returns**

Whether it's value was set as the default.

**0.15.486.3.4 `load()`** `bool delta::Settings::load (`  
    `int argc,`  
    `char * argv[] ) [inline]`

Load options from command line.

**Parameters**

|             |                       |
|-------------|-----------------------|
| <i>argc</i> | Number of arguments.  |
| <i>argv</i> | Content of arguments. |

**0.15.486.3.5 set()** template<typename T >

```
void delta::Settings::set (
 const std::string & s,
 const T & t) [inline]
```

Set option with given name as a given type.

If there is no such option exceptions may be thrown.

**Parameters**

|          |                     |
|----------|---------------------|
| <i>s</i> | Name of the option. |
| <i>t</i> | New value.          |

**0.15.486.4 Friends And Related Function Documentation****0.15.486.4.1 operator<<** std::ostream& operator<< (

```
 std::ostream & os,
 const Settings & s) [friend]
```

Streaming operator.

**Parameters**

|           |                |
|-----------|----------------|
| <i>os</i> | Output stream. |
| <i>s</i>  | Settings.      |

**Returns**

os.

**0.15.487 smtrat::settings::Settings Struct Reference**

```
#include <Settings.h>
```

**0.15.488 smtrat::SettingsComponents Class Reference**

```
#include <SettingsComponents.h>
```

**Public Member Functions**

- void [add](#) (std::function< void(SettingsParser &)> &&f)
- void [add\\_to\\_parser](#) (SettingsParser &parser) const

**0.15.488.1 Member Function Documentation**

```
0.15.488.1.1 add() void smtrat::SettingsComponents::add (std::function< void(SettingsParser &) > && f) [inline]
```

```
0.15.488.1.2 add_to_parser() void smtrat::SettingsComponents::add_to_parser (SettingsParser & parser) const [inline]
```

## 0.15.489 **benchmax::SettingsParser Class Reference**

Generic class to manage settings parsing using boost::program\_options.

```
#include <SettingsParser.h>
```

### 0.15.489.1 Detailed Description

Generic class to manage settings parsing using boost::program\_options.

Allows to register dynamically add new options\_description object and manages parsing them from command line and config file. When everything is registered finalize() has to be called to construct the full option description.

## 0.15.490 **smtrat::SettingsParser Class Reference**

```
#include <SettingsParser.h>
```

## 0.15.491 **smtrat::CNFerModule::SettingsType Struct Reference**

```
#include <CNFerModule.h>
```

### Static Public Attributes

- static constexpr auto moduleName = "CNFerModule"

### 0.15.491.1 Field Documentation

```
0.15.491.1.1 moduleName constexpr auto smtrat::CNFerModule::SettingsType::moduleName = "CNFerModule" [static], [constexpr]
```

## 0.15.492 **smtrat::parser::SExpressionParser Struct Reference**

```
#include <SExpression.h>
```

### Public Member Functions

- [SExpressionParser \(\)](#)

### Data Fields

- [SpecConstantParser speconstant](#)
- [SymbolParser symbol](#)
- [KeywordParser keyword](#)
- [qi::rule< Iterator, SExpression< types::ConstType >, Skipper > main](#)

### 0.15.492.1 Constructor & Destructor Documentation

```
0.15.492.1.1 SExpressionParser() smtrat::parser::SExpressionParser::SExpressionParser () [inline]
```

### 0.15.492.2 Field Documentation

**0.15.492.2.1 keyword** `KeywordParser smtrat::parser::SExpressionParser::keyword`

**0.15.492.2.2 main** `qi::rule<Iterator, SExpression<types::ConstType>, Skipper> smtrat::parser::SExpressionParser::main`

**0.15.492.2.3 speccconstant** `SpecConstantParser smtrat::parser::SExpressionParser::speccconstant`

**0.15.492.2.4 symbol** `SymbolParser smtrat::parser::SExpressionParser::symbol`

## 0.15.493 smtrat::parser::SExpressionSequence< T > Struct Template Reference

#include <Common.h>

### Public Member Functions

- `SExpressionSequence (const std::vector< SExpression< T >> &v)`
- `SExpressionSequence (std::vector< SExpression< T >> &&v)`

### Data Fields

- **T elements**

*STL member.*

### 0.15.493.1 Constructor & Destructor Documentation

**0.15.493.1.1 SExpressionSequence()** [1/2] `template<typename T > smtrat::parser::SExpressionSequence< T >::SExpressionSequence ( const std::vector< SExpression< T >> & v ) [inline]`

**0.15.493.1.2 SExpressionSequence()** [2/2] `template<typename T > smtrat::parser::SExpressionSequence< T >::SExpressionSequence ( std::vector< SExpression< T >> && v ) [inline]`

### 0.15.493.2 Field Documentation

**0.15.493.2.1 elements** `T std::vector< T >::elements [inherited]`  
STL member.

## 0.15.494 smtrat::ShortFormulaEncoder Class Reference

#include <ShortFormulaEncoder.h>

## Public Member Functions

- `ShortFormulaEncoder ()`
- `bool canEncode (const ConstraintT &constraint)`
- `Rational encodingSize (const ConstraintT &constraint)`
- `std::string name ()`
- `std::optional< FormulaT > encode (const ConstraintT &constraint)`

*Encodes an arbitrary constraint.*

## Data Fields

- `std::size_t problem_size`

## Protected Member Functions

- `std::optional< FormulaT > doEncode (const ConstraintT &constraint)`
- `FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)`

### 0.15.494.1 Constructor & Destructor Documentation

#### 0.15.494.1.1 `ShortFormulaEncoder()` `smtrat::ShortFormulaEncoder::ShortFormulaEncoder ( ) [inline]`

#### 0.15.494.2 Member Function Documentation

##### 0.15.494.2.1 `canEncode()` `bool smtrat::ShortFormulaEncoder::canEncode (` `const ConstraintT & constraint ) [virtual]` Implements [smtrat::PseudoBoolEncoder](#).

##### 0.15.494.2.2 `doEncode()` `std::optional< FormulaT > smtrat::ShortFormulaEncoder::doEncode (` `const ConstraintT & constraint ) [protected], [virtual]` Implements [smtrat::PseudoBoolEncoder](#).

##### 0.15.494.2.3 `encode()` `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (` `const ConstraintT & constraint ) [inherited]` Encodes an arbitrary constraint.

#### Returns

encoded formula

##### 0.15.494.2.4 `encodingSize()` `Rational smtrat::ShortFormulaEncoder::encodingSize (` `const ConstraintT & constraint ) [virtual]` Reimplemented from [smtrat::PseudoBoolEncoder](#).

##### 0.15.494.2.5 `generateVarChain()` `FormulaT smtrat::PseudoBoolEncoder::generateVarChain (` `const std::set< carl::Variable > & vars,` `carl::FormulaType type ) [protected], [inherited]`

**0.15.494.2.6 name()** std::string smtrat::ShortFormulaEncoder::name ( ) [inline], [virtual]  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

### 0.15.494.3 Field Documentation

**0.15.494.3.1 problem\_size** std::size\_t smtrat::PseudoBoolEncoder::problem\_size [inherited]

## 0.15.495 benchmax::simple\_parser Class Reference

```
#include <SMTRAT.h>
```

### Public Member Functions

- [simple\\_parser](#) (const std::string &content)
- [bool expect](#) (const char c)
- [bool expect\\_end](#) ()
- [void skip\\_whitespace](#) ()
- [void skip\\_excluding](#) (const char c)
- [std::optional< std::string > read\\_until](#) (const char c)
- [std::optional< std::string > read\\_until\\_whitespace](#) ()
- [std::optional< std::string > read\\_until\\_whitespace\\_or](#) (const std::initializer\_list< char > cs)

### 0.15.495.1 Constructor & Destructor Documentation

**0.15.495.1.1 simple\_parser()** benchmax::simple\_parser::simple\_parser (const std::string & content) [inline]

### 0.15.495.2 Member Function Documentation

**0.15.495.2.1 expect()** bool benchmax::simple\_parser::expect (const char c) [inline]

**0.15.495.2.2 expect\_end()** bool benchmax::simple\_parser::expect\_end () [inline]

**0.15.495.2.3 read\_until()** std::optional<std::string> benchmax::simple\_parser::read\_until (const char c) [inline]

**0.15.495.2.4 read\_until\_whitespace()** std::optional<std::string> benchmax::simple\_parser::read\_until\_whitespace () [inline]

**0.15.495.2.5 read\_until\_whitespace\_or()** std::optional<std::string> benchmax::simple\_parser::read\_until\_whitespace\_or (const std::initializer\_list< char > cs) [inline]

```
0.15.495.2.6 skip_excluding() void benchmax::simple_parser::skip_excluding (
 const char c) [inline]
```

```
0.15.495.2.7 skip_whitespace() void benchmax::simple_parser::skip_whitespace () [inline]
```

## 0.15.496 smtrat::parser::Theories::SimpleSortAdder Struct Reference

Helper functor for [addSimpleSorts\(\)](#) method.

```
#include <Theories.h>
```

### Public Member Functions

- [SimpleSortAdder](#) (qi::symbols< char, carl::Sort > & **sorts**)
- template<typename T >  
void [operator\(\)](#) (T \*)

### Data Fields

- qi::symbols< char, carl::Sort > & **sorts**

### 0.15.496.1 Detailed Description

Helper functor for [addSimpleSorts\(\)](#) method.

### 0.15.496.2 Constructor & Destructor Documentation

```
0.15.496.2.1 SimpleSortAdder() smtrat::parser::Theories::SimpleSortAdder (
 qi::symbols< char, carl::Sort > & sorts) [inline]
```

### 0.15.496.3 Member Function Documentation

```
0.15.496.3.1 operator()() template<typename T >
void smtrat::parser::Theories::SimpleSortAdder::operator() (
 T *) [inline]
```

### 0.15.496.4 Field Documentation

```
0.15.496.4.1 sorts qi::symbols<char, carl::Sort>& smtrat::parser::Theories::SimpleSortAdder::sorts
```

## 0.15.497 smtrat::parser::SimpleSymbolParser Struct Reference

Parses symbols: simple\_symbol | quoted\_symbol where.

```
#include <Lexicon.h>
```

### Public Member Functions

- [SimpleSymbolParser](#) ()

### Data Fields

- qi::rule< [Iterator](#), std::string(), [Skipper](#) > **main**

### 0.15.497.1 Detailed Description

Parses symbols: simple\_symbol | quoted\_symbol where.

- simple\_symbol is any string of [0-9a-zA-Z~!@#\$%^&\*\_+=<>.?/] that does not start with a digit and is not a reserved word.
- quoted\_symbol is any string of printable characters (including space, tab, line-breaks) except \ and | enclosed in | characters.

### 0.15.497.2 Constructor & Destructor Documentation

**0.15.497.2.1 SimpleSymbolParser()** smtrat::parser::SimpleSymbolParser::SimpleSymbolParser ( )  
[inline]

### 0.15.497.3 Field Documentation

**0.15.497.3.1 main** qi::rule<[Iterator](#), std::string(), [Skipper](#)> smtrat::parser::SimpleSymbolParser::main

## 0.15.498 smtrat::expression::simplifier::Simplifier Class Reference

#include <Simplifier.h>

### Public Member Functions

- const [ExpressionContent \\* operator\(\)](#) (const [ExpressionContent \\* \\_ec](#)) const

### 0.15.498.1 Member Function Documentation

**0.15.498.1.1 operator()** const [ExpressionContent \\* smtrat::expression::simplifier::Simplifier::operator\(\)](#) (const [ExpressionContent \\* \\_ec](#)) const [inline]

## 0.15.499 smtrat::expression::simplifier::SimplifierChainCaller< chainID > Struct Template Reference

#include <Simplifier.h>

### Public Member Functions

- const [ExpressionContent \\* operator\(\)](#) (const [ExpressionContent \\* \\_ec](#), const [SimplifierChain & \\_chain](#)) const

### Data Fields

- [SimplifierChainCaller< chainID-1 > recurse](#)

### 0.15.499.1 Member Function Documentation

```
0.15.499.1.1 operator() template<std::size_t chainID = std::tuple_size<SimplifierChain>::value - 1>
const ExpressionContent* smrat::expression::simplifier::SimplifierChainCaller< chainID >::operator() (
 const ExpressionContent * _ec,
 const SimplifierChain & _chain) const [inline]
```

## 0.15.499.2 Field Documentation

```
0.15.499.2.1 recurse template<std::size_t chainID = std::tuple_size<SimplifierChain>::value - 1>
SimplifierChainCaller<chainID-1> smrat::expression::simplifier::SimplifierChainCaller< chainID >::recurse
```

## 0.15.500 smrat::expression::simplifier::SimplifierChainCaller< 0 > Struct Reference

```
#include <Simplifier.h>
```

### Public Member Functions

- const ExpressionContent \* **operator()** (const ExpressionContent \* \_ec, const SimplifierChain & \_chain) const

#### 0.15.500.1 Member Function Documentation

```
0.15.500.1.1 operator() const ExpressionContent* smrat::expression::simplifier::SimplifierChainCaller< 0 >::operator() (
 const ExpressionContent * _ec,
 const SimplifierChain & _chain) const [inline]
```

## 0.15.501 smrat::expression::simplifier::SingletonSimplifier Struct Reference

```
#include <SingletonSimplifier.h>
```

### Public Member Functions

- const ExpressionContent \* **simplify** (const NaryExpression & expr) const
- const ExpressionContent \* **operator()** (const carl::Variable & expr) const
- const ExpressionContent \* **operator()** (const ITEExpression & expr) const
- const ExpressionContent \* **operator()** (const QuantifierExpression & expr) const
- const ExpressionContent \* **operator()** (const UnaryExpression & expr) const
- const ExpressionContent \* **operator()** (const BinaryExpression & expr) const
- const ExpressionContent \* **operator()** (const NaryExpression & expr) const
- const ExpressionContent \* **operator()** (const ExpressionContent \* \_ec) const

### Protected Member Functions

- virtual const ExpressionContent \* **simplify** (const carl::Variable &) const
- virtual const ExpressionContent \* **simplify** (const ITEExpression &) const
- virtual const ExpressionContent \* **simplify** (const QuantifierExpression &) const
- virtual const ExpressionContent \* **simplify** (const UnaryExpression &) const
- virtual const ExpressionContent \* **simplify** (const BinaryExpression &) const

### 0.15.501.1 Member Function Documentation

**0.15.501.1.1 operator() [1/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::operator() (

const `BinaryExpression & expr` ) const [inline], [inherited]

**0.15.501.1.2 operator() [2/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::operator() (

const `carl::Variable & expr` ) const [inline], [inherited]

**0.15.501.1.3 operator() [3/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::operator() (

const `ExpressionContent * _ec` ) const [inline], [inherited]

**0.15.501.1.4 operator() [4/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::operator() (

const `ITEExpression & expr` ) const [inline], [inherited]

**0.15.501.1.5 operator() [5/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::operator() (

const `NaryExpression & expr` ) const [inline], [inherited]

**0.15.501.1.6 operator() [6/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::operator() (

const `QuantifierExpression & expr` ) const [inline], [inherited]

**0.15.501.1.7 operator() [7/7]** const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::operator() (

const `UnaryExpression & expr` ) const [inline], [inherited]

**0.15.501.1.8 simplify() [1/6]** virtual const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::simplify (

const `BinaryExpression &`  ) const [inline], [protected], [virtual], [inherited]

**0.15.501.1.9 simplify() [2/6]** virtual const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::simplify (

const `carl::Variable &`  ) const [inline], [protected], [virtual], [inherited]

**0.15.501.1.10 simplify() [3/6]** virtual const `ExpressionContent*` smrat::expression::simplifier::Base<→  
Simplifier::simplify (

const `ITEExpression &`  ) const [inline], [protected], [virtual], [inherited]

**0.15.501.1.11 `simplify()` [4/6]** const `ExpressionContent*` smtrat::expression::simplifier::Singleton<→  
Simplifier::simplify (  
    const `NaryExpression` & `expr`) const [inline], [virtual]  
Reimplemented from `smtrat::expression::simplifier::BaseSimplifier`.

**0.15.501.1.12 `simplify()` [5/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier<→  
::BaseSimplifier::simplify (  
    const `QuantifierExpression` & ) const [inline], [protected], [virtual], [inherited]

**0.15.501.1.13 `simplify()` [6/6]** virtual const `ExpressionContent*` smtrat::expression::simplifier<→  
::BaseSimplifier::simplify (  
    const `UnaryExpression` & ) const [inline], [protected], [virtual], [inherited]  
Reimplemented in `smtrat::expression::simplifier::NegationSimplifier`.

## 0.15.502 `smtrat::cad::sample_compare::size` Struct Reference

```
#include <SampleComparator.h>
```

### 0.15.503 `delta::Skipper` Struct Reference

This class is a `boost::spirit::qi` grammar that matches whitespaces and smtlib comments.  
`#include <Parser.h>`

#### Public Member Functions

- `Skipper()`

#### Data Fields

- `qi::rule<Iterator> main`

#### 0.15.503.1 Detailed Description

This class is a `boost::spirit::qi` grammar that matches whitespaces and smtlib comments.

#### 0.15.503.2 Constructor & Destructor Documentation

##### 0.15.503.2.1 `Skipper()` `delta::Skipper::Skipper ()` [inline]

##### 0.15.503.3 Field Documentation

##### 0.15.503.3.1 `main` `qi::rule<Iterator> delta::Skipper::main`

## 0.15.504 `smtrat::parser::Skipper` Struct Reference

```
#include <Common.h>
```

#### Public Member Functions

- `Skipper()`

## Data Fields

- boost::spirit::qi::rule<[Iterator](#)> `main`

### 0.15.504.1 Constructor & Destructor Documentation

#### 0.15.504.1.1 `Skipper()` smtrat::parser::Skipper::Skipper ( ) [inline]

### 0.15.504.2 Field Documentation

#### 0.15.504.2.1 `main` boost::spirit::qi::rule<[Iterator](#)> smtrat::parser::Skipper::main

## 0.15.505 `benchmax::SlurmBackend` Class Reference

[Backend](#) for the Slurm workload manager.

```
#include <SlurmBackend.h>
```

### Public Member Functions

- bool `suspendable` () const
  - Run all tools on all benchmarks using Slurm.*
- void `run` (const [Jobs](#) &jobs, bool `wait_for_termination`)
  - Add a result.*
- void `process_results` (const [Jobs](#) &jobs, bool `check_finished`)
- void `addResult` (const [Tool](#) \*tool, const [fs::path](#) &file, [BenchmarkResult](#) &&result)
  - Add a result.*

### Protected Member Functions

- virtual void `startTool` (const [Tool](#) \*)
  - Hook for every tool at the beginning.*
- virtual void `finalize` ()
  - Hook to allow for asynchronous backends to wait for jobs to terminate.*
- virtual void `execute` (const [Tool](#) \*, const [fs::path](#) &)
  - Execute a single pair of tool and benchmark.*
- void `madeProgress` ([std::size\\_t](#) files=1)
  - Can be called to give information about the current progress, if available.*
- void `sanitize_results` (const [Jobs](#) &jobs) const
- void `write_results` (const [Jobs](#) &jobs) const

### Protected Attributes

- [std::size\\_t](#) `mExpectedJobs`
  - Number of jobs that should be run.*
- [std::atomic< std::size\\_t >](#) `mFinishedJobs`
  - Number of jobs that are finished.*
- [std::atomic< std::size\\_t >](#) `mLastPercent`
  - Percentage of finished jobs when `madeProgress()` was last called.*

### 0.15.505.1 Detailed Description

[Backend](#) for the Slurm workload manager.

The execution model is as follows: We create multiple jobs that each consists of multiple array jobs that each execute one slice of the task list. One array job executes `Settings::slice_size` entries of the task list. One job consists of `Settings::array_size` array jobs. We start as many jobs as necessary.

### 0.15.505.2 Member Function Documentation

**0.15.505.2.1 addResult()** void benchmax::Backend::addResult ( const [Tool](#) \* *tool*, const [fs::path](#) & *file*, [BenchmarkResult](#) && *result* ) [inline], [inherited]

Add a result.

**0.15.505.2.2 execute()** virtual void benchmax::Backend::execute ( const [Tool](#) \* , const [fs::path](#) & ) [inline], [protected], [virtual], [inherited]

Execute a single pair of tool and benchmark.

Reimplemented in [benchmax::LocalBackend](#), and [benchmax::CondorBackend](#).

**0.15.505.2.3 finalize()** virtual void benchmax::Backend::finalize ( ) [inline], [protected], [virtual], [inherited]

Hook to allow for asynchronous backends to wait for jobs to terminate.

**0.15.505.2.4 madeProgress()** void benchmax::Backend::madeProgress ( std::size\_t *files* = 1 ) [inline], [protected], [inherited]

Can be called to give information about the current progress, if available.

**0.15.505.2.5 process\_results()** void benchmax::Backend::process\_results ( const [Jobs](#) & *jobs*, bool *check\_finished* ) [inline], [inherited]

**0.15.505.2.6 run()** void benchmax::SlurmBackend::run ( const [Jobs](#) & *jobs*, bool *wait\_for\_termination* ) [inline]

Run all tools on all benchmarks using Slurm.

**0.15.505.2.7 sanitize\_results()** void benchmax::Backend::sanitize\_results ( const [Jobs](#) & *jobs* ) const [inline], [protected], [inherited]

**0.15.505.2.8 startTool()** virtual void benchmax::Backend::startTool ( const [Tool](#) \* ) [inline], [protected], [virtual], [inherited]

Hook for every tool at the beginning.

Can be used to upload the tool to some remote system.

**0.15.505.2.9 suspendable()** bool benchmax::SlurmBackend::suspendable ( ) const [inline]

**0.15.505.2.10 write\_results()** void benchmax::Backend::write\_results ( const [Jobs](#) & *jobs* ) const [inline], [protected], [inherited]

### 0.15.505.3 Field Documentation

**0.15.505.3.1 mExpectedJobs** std::size\_t benchmax::Backend::mExpectedJobs [protected], [inherited]  
Number of jobs that should be run.

**0.15.505.3.2 mFinishedJobs** std::atomic<std::size\_t> benchmax::Backend::mFinishedJobs [protected], [inherited]  
Number of jobs that are finished.

**0.15.505.3.3 mLastPercent** std::atomic<std::size\_t> benchmax::Backend::mLastPercent [protected], [inherited]  
Percentage of finished jobs when [madeProgress\(\)](#) was last called.

## 0.15.506 benchmax::settings::SlurmBackendSettings Struct Reference

[Settings](#) for the Slurm backend.

```
#include <SlurmSettings.h>
```

### Data Fields

- std::size\_t [array\\_size](#)  
*Number of array jobs within one job.*
- std::size\_t [slice\\_size](#)  
*Size of one slice that is handled in one array job.*
- std::string [tmp\\_dir](#)  
*Temporary directory for output files.*
- bool [keep\\_logs](#)  
*Do not remove logs from file system if set to true.*
- std::string [archive\\_log\\_file](#)  
*Puts logs to some archive.*
- std::string [sbatch\\_options](#)  
*Additional options passed on to slurm.*
- carl::settings::duration [submission\\_delay](#)  
*Delay between job submissions.*

### 0.15.506.1 Detailed Description

[Settings](#) for the Slurm backend.

### 0.15.506.2 Field Documentation

**0.15.506.2.1 archive\_log\_file** std::string benchmax::settings::SlurmBackendSettings::archive\_↔  
log\_file  
Puts logs to some archive.

**0.15.506.2.2 array\_size** std::size\_t benchmax::settings::SlurmBackendSettings::array\_size  
Number of array jobs within one job.

**0.15.506.2.3 keep\_logs** bool benchmax::settings::SlurmBackendSettings::keep\_logs  
Do not remove logs from file system if set to true.

**0.15.506.2.4 sbatch\_options** std::string benchmax::settings::SlurmBackendSettings::sbatch\_↔  
options  
Additional options passed on to slurm.

**0.15.506.2.5 slice\_size** std::size\_t benchmax::settings::SlurmBackendSettings::slice\_size  
Size of one slice that is handled in one array job.

**0.15.506.2.6 submission\_delay** carl::settings::duration benchmax::settings::SlurmBackend↔  
Settings::submission\_delay  
Delay between job submissions.

**0.15.506.2.7 tmp\_dir** std::string benchmax::settings::SlurmBackendSettings::tmp\_dir  
Temporary directory for output files.

## 0.15.507 smrat::parser::SMTLIBParser Class Reference

```
#include <Parser.h>
```

### Public Member Functions

- [SMTLIBParser \(InstructionHandler &handler, bool queueInstructions\)](#)
- [~SMTLIBParser \(\)](#)
- [bool parse \(std::istream &in\)](#)
- [void add \(const types::TermType &t, bool isSoftFormula=false, Rational weight=Rational\(1\), const std::string id=std::string\(\)\)](#)
- [void check \(\)](#)
- [void declareConst \(const std::string &name, const carl::Sort &sort\)](#)
- [void declareFun \(const std::string &name, const std::vector< carl::Sort > &args, const carl::Sort &sort\)](#)
- [void declareSort \(const std::string &name, Integer arity\)](#)
- [void echo \(const std::string &s\)](#)
- [void eliminateQuantifiers \(const qe::QEQuery &q\)](#)
- [void exit \(\)](#)
- [void getAllModels \(\)](#)
- [void getAssertions \(\)](#)
- [void getAssignment \(\)](#)
- [void getInfo \(const std::string &key\)](#)
- [void getModel \(\)](#)
- [void getObjectives \(\)](#)
- [void getOption \(const std::string &key\)](#)
- [void getProof \(\)](#)
- [void getUnsatCore \(\)](#)
- [void getValue \(\[\[maybe\\_unused\]\] const std::vector< types::TermType > &vars\)](#)
- [void addObjective \(const types::TermType &t, OptimizationType ot\)](#)
- [void pop \(const Integer &n\)](#)
- [void push \(const Integer &n\)](#)
- [void reset \(\)](#)
- [void resetAssertions \(\)](#)
- [void setInfo \(const Attribute &attribute\)](#)
- [void setLogic \(const carl::Logic &name\)](#)
- [void setOption \(const Attribute &option\)](#)
- [template<typename Function , typename... Args>  
void callHandler \(const Function &f, const Args &... args\)](#)

**Data Fields**

- bool `queueInstructions`
- `InstructionHandler & handler`
- `ParserState state`
- `Theories theories`
- `ScriptParser< SMTLIBParser > parser`

**0.15.507.1 Constructor & Destructor Documentation**

**0.15.507.1.1 `SMTLIBParser()`** `smtrat::parser::SMTLIBParser::SMTLIBParser (`  
`InstructionHandler & handler,`  
`bool queueInstructions )` [inline]

**0.15.507.1.2 `~SMTLIBParser()`** `smtrat::parser::SMTLIBParser::~SMTLIBParser ( )` [inline]

**0.15.507.2 Member Function Documentation**

**0.15.507.2.1 `add()`** `void smtrat::parser::SMTLIBParser::add (`  
`const types::TermType & t,`  
`bool isSoftFormula = false,`  
`Rational weight = Rational(1),`  
`const std::string id = std::string() )` [inline]

**0.15.507.2.2 `addObjective()`** `void smtrat::parser::SMTLIBParser::addObjective (`  
`const types::TermType & t,`  
`OptimizationType ot )` [inline]

**0.15.507.2.3 `callHandler()`** `template<typename Function , typename... Args>`  
`void smtrat::parser::SMTLIBParser::callHandler (`  
`const Function & f,`  
`const Args &... args )` [inline]

**0.15.507.2.4 `check()`** `void smtrat::parser::SMTLIBParser::check ( )` [inline]

**0.15.507.2.5 `declareConst()`** `void smtrat::parser::SMTLIBParser::declareConst (`  
`const std::string & name,`  
`const carl::Sort & sort )` [inline]

**0.15.507.2.6 `declareFun()`** `void smtrat::parser::SMTLIBParser::declareFun (`  
`const std::string & name,`  
`const std::vector< carl::Sort > & args,`  
`const carl::Sort & sort )` [inline]

**0.15.507.2.7 declareSort()** void smtrat::parser::SMTLIBParser::declareSort ( const std::string & name, Integer arity ) [inline]

**0.15.507.2.8 echo()** void smtrat::parser::SMTLIBParser::echo ( const std::string & s ) [inline]

**0.15.507.2.9 eliminateQuantifiers()** void smtrat::parser::SMTLIBParser::eliminateQuantifiers ( const qe::QEQuery & q ) [inline]

**0.15.507.2.10 exit()** void smtrat::parser::SMTLIBParser::exit ( ) [inline]

**0.15.507.2.11 getAllModels()** void smtrat::parser::SMTLIBParser::getAllModels ( ) [inline]

**0.15.507.2.12 getAssertions()** void smtrat::parser::SMTLIBParser::getAssertions ( ) [inline]

**0.15.507.2.13 getAssignment()** void smtrat::parser::SMTLIBParser::getAssignment ( ) [inline]

**0.15.507.2.14 getInfo()** void smtrat::parser::SMTLIBParser::getInfo ( const std::string & key ) [inline]

**0.15.507.2.15 getModel()** void smtrat::parser::SMTLIBParser::getModel ( ) [inline]

**0.15.507.2.16 getObjectives()** void smtrat::parser::SMTLIBParser::getObjectives ( ) [inline]

**0.15.507.2.17 getOption()** void smtrat::parser::SMTLIBParser::getOption ( const std::string & key ) [inline]

**0.15.507.2.18 getProof()** void smtrat::parser::SMTLIBParser::getProof ( ) [inline]

**0.15.507.2.19 getUnsatCore()** void smtrat::parser::SMTLIBParser::getUnsatCore ( ) [inline]

**0.15.507.2.20 getValue()** void smtrat::parser::SMTLIBParser::getValue ( [[maybe\_unused]] const std::vector< types::TermType > & vars ) [inline]

**0.15.507.2.21 parse()** bool smtrat::parser::SMTLIBParser::parse ( std::istream & in ) [inline]

**0.15.507.2.22** **pop()** void smtrat::parser::SMTLIBParser::pop ( const Integer & n ) [inline]

**0.15.507.2.23** **push()** void smtrat::parser::SMTLIBParser::push ( const Integer & n ) [inline]

**0.15.507.2.24** **reset()** void smtrat::parser::SMTLIBParser::reset ( ) [inline]

**0.15.507.2.25** **resetAssertions()** void smtrat::parser::SMTLIBParser::resetAssertions ( ) [inline]

**0.15.507.2.26** **setInfo()** void smtrat::parser::SMTLIBParser::setInfo ( const Attribute & attribute ) [inline]

**0.15.507.2.27** **setLogic()** void smtrat::parser::SMTLIBParser::setLogic ( const carl::Logic & name ) [inline]

**0.15.507.2.28** **setOption()** void smtrat::parser::SMTLIBParser::setOption ( const Attribute & option ) [inline]

### 0.15.507.3 Field Documentation

**0.15.507.3.1** **handler** InstructionHandler& smtrat::parser::SMTLIBParser::handler

**0.15.507.3.2** **parser** ScriptParser<SMTLIBParser> smtrat::parser::SMTLIBParser::parser

**0.15.507.3.3** **queueInstructions** bool smtrat::parser::SMTLIBParser::queueInstructions

**0.15.507.3.4** **state** ParserState smtrat::parser::SMTLIBParser::state

**0.15.507.3.5** **theories** Theories smtrat::parser::SMTLIBParser::theories

## 0.15.508 benchmax::SMTRAT Class Reference

Tool wrapper for SMT-RAT for SMT-LIB.

```
#include <SMTRAT.h>
```

### Public Member Functions

- **SMTRAT** (const fs::path &**binary**, const std::string &**arguments**)  
*New SMT-RAT tool.*
- virtual bool **canHandle** (const fs::path &**path**) const override  
*Only handle .smt2 files.*
- std::string **getStatusFromOutput** (const BenchmarkResult &**result**) const

- Try to parse memouts from stderr.
- bool `parse_stats` (`BenchmarkResult` &`result`) const
- virtual void `additionalResults` (const `fs::path` &, `BenchmarkResult` &`result`) const override
 

Computes the answer string from the exit code and parses statistics output.
- `std::string name` () const
 

Common name of this tool.
- `fs::path binary` () const
 

Full path to the binary.
- const `std::map< std::string, std::string >` & `attributes` () const
 

A set of attributes, for example compilation options.
- `std::vector< std::string >` `resolveDependencies` () const
 

Get dependencies of binary required to run it (via ldd)
- `std::size_t attributeHash` () const
 

Hash of the attributes.
- virtual `std::string getCommandline` (const `std::string &file`) const
 

Compose commandline for this tool and the given input file.
- virtual `std::string getCommandline` (const `std::string &file`, const `std::string &localBinary`) const
 

Compose commandline for this tool with another binary name and the given input file.
- virtual `std::optional< std::string >` `parseCommandline` (const `std::string &cmdline`) const
 

Compose commandline for this tool and the given input file.

## Protected Attributes

- `std::string mName`

(Non-unique) identifier for the tool.
- `fs::path mBinary`

Path to the binary.
- `std::string mArguments`

Command line arguments that should be passed to the binary.
- `std::map< std::string, std::string >` `mAttributes`

Attributes of the tool obtained by introspection of the binary.

### 0.15.508.1 Detailed Description

`Tool` wrapper for SMT-RAT for SMT-LIB.

### 0.15.508.2 Constructor & Destructor Documentation

```
0.15.508.2.1 SMTRAT() benchmax::SMTRAT::SMTRAT (
 const fs::path & binary,
 const std::string & arguments) [inline]
```

New SMT-RAT tool.

### 0.15.508.3 Member Function Documentation

```
0.15.508.3.1 additionalResults() virtual void benchmax::SMTRAT::additionalResults (
 const fs::path & ,
 BenchmarkResult & result) const [inline], [override], [virtual]
```

Computes the answer string from the exit code and parses statistics output.

Reimplemented from `benchmax::Tool`.

**0.15.508.3.2 attributeHash()** `std::size_t benchmax::Tool::attributeHash () const [inline], [inherited]`  
Hash of the attributes.

**0.15.508.3.3 attributes()** `const std::map<std::string, std::string>& benchmax::Tool::attributes () const [inline], [inherited]`  
A set of attributes, for example compilation options.

**0.15.508.3.4 binary()** `fs::path benchmax::Tool::binary () const [inline], [inherited]`  
Full path to the binary.

**0.15.508.3.5 canHandle()** `virtual bool benchmax::SMTRAT::canHandle (const fs::path & path) const [inline], [override], [virtual]`  
Only handle .smt2 files.  
Reimplemented from [benchmax::Tool](#).  
Reimplemented in [benchmax::SMTRAT\\_OPB](#).

**0.15.508.3.6 getCommandline() [1/2]** `virtual std::string benchmax::Tool::getCommandline (const std::string & file) const [inline], [virtual], [inherited]`  
Compose commandline for this tool and the given input file.

**0.15.508.3.7 getCommandline() [2/2]** `virtual std::string benchmax::Tool::getCommandline (const std::string & file, const std::string & localBinary) const [inline], [virtual], [inherited]`  
Compose commandline for this tool with another binary name and the given input file.

**0.15.508.3.8 getStatusFromOutput()** `std::string benchmax::SMTRAT::getStatusFromOutput (const BenchmarkResult & result) const [inline]`  
Try to parse memouts from stderr.

**0.15.508.3.9 name()** `std::string benchmax::Tool::name () const [inline], [inherited]`  
Common name of this tool.

**0.15.508.3.10 parse\_stats()** `bool benchmax::SMTRAT::parse_stats (BenchmarkResult & result) const [inline]`

**0.15.508.3.11 parseCommandLine()** `virtual std::optional<std::string> benchmax::Tool::parseCommandline (const std::string & cmdline) const [inline], [virtual], [inherited]`  
Compose commandline for this tool and the given input file.

**0.15.508.3.12 resolveDependencies()** `std::vector<std::string> benchmax::Tool::resolveDependencies () const [inline], [inherited]`  
Get dependencies of binary required to run it (via ldd)

#### 0.15.508.4 Field Documentation

**0.15.508.4.1 mArguments** std::string benchmax::Tool::mArguments [protected], [inherited]  
Command line arguments that should be passed to the binary.

**0.15.508.4.2 mAttributes** std::map<std::string, std::string> benchmax::Tool::mAttributes [protected], [inherited]  
Attributes of the tool obtained by introspection of the binary.

**0.15.508.4.3 mBinary** fs::path benchmax::Tool::mBinary [protected], [inherited]  
Path to the binary.

**0.15.508.4.4 mName** std::string benchmax::Tool::mName [protected], [inherited]  
(Non-unique) identifier for the tool.

### 0.15.509 benchmax::SMTRAT\_Analyzer Class Reference

Tool wrapper for SMT-RAT for SMT-LIB.

```
#include <SMTRAT_Analyzer.h>
```

#### Public Member Functions

- **SMTRAT\_Analyzer** (const fs::path &**binary**, const std::string &**arguments**)  
*New SMT-RAT tool.*
- virtual bool **canHandle** (const fs::path &**path**) const override  
*Only handle .smt2 files.*
- virtual void **additionalResults** (const fs::path &, **BenchmarkResult** &**result**) const override  
*Computes the answer string from the exit code and parses statistics output.*
- std::string **name** () const  
*Common name of this tool.*
- fs::path **binary** () const  
*Full path to the binary.*
- const std::map<std::string, std::string> &**attributes** () const  
*A set of attributes, for example compilation options.*
- std::vector<std::string> **resolveDependencies** () const  
*Get dependencies of binary required to run it (via ldd)*
- std::size\_t **attributeHash** () const  
*Hash of the attributes.*
- virtual std::string **getCommandline** (const std::string &**file**) const  
*Compose commandline for this tool and the given input file.*
- virtual std::string **getCommandline** (const std::string &**file**, const std::string &**localBinary**) const  
*Compose commandline for this tool with another binary name and the given input file.*
- virtual std::optional<std::string> **parseCommandline** (const std::string &**cmdline**) const  
*Compose commandline for this tool and the given input file.*

## Protected Attributes

- std::string **mName**  
*(Non-unique) identifier for the tool.*
- fs::path **mBinary**  
*Path to the binary.*
- std::string **mArguments**  
*Command line arguments that should be passed to the binary.*
- std::map< std::string, std::string > **mAttributes**  
*Attributes of the tool obtained by introspection of the binary.*

### 0.15.509.1 Detailed Description

Tool wrapper for SMT-RAT for SMT-LIB.

### 0.15.509.2 Constructor & Destructor Documentation

```
0.15.509.2.1 SMTRAT_Analyzer() benchmax::SMTRAT_Analyzer::SMTRAT_Analyzer (
 const fs::path & binary,
 const std::string & arguments) [inline]
```

New SMT-RAT tool.

### 0.15.509.3 Member Function Documentation

```
0.15.509.3.1 additionalResults() virtual void benchmax::SMTRAT_Analyzer::additionalResults (
 const fs::path & ,
 BenchmarkResult & result) const [inline], [override], [virtual]
```

Computes the answer string from the exit code and parses statistics output.

Reimplemented from [benchmax::Tool](#).

```
0.15.509.3.2 attributeHash() std::size_t benchmax::Tool::attributeHash () const [inline],
[inherited]
```

Hash of the attributes.

```
0.15.509.3.3 attributes() const std::map<std::string, std::string>& benchmax::Tool::attributes (
) const [inline], [inherited]
```

A set of attributes, for example compilation options.

```
0.15.509.3.4 binary() fs::path benchmax::Tool::binary () const [inline], [inherited]
```

Full path to the binary.

```
0.15.509.3.5 canHandle() virtual bool benchmax::SMTRAT_Analyzer::canHandle (
 const fs::path & path) const [inline], [override], [virtual]
```

Only handle .smt2 files.

Reimplemented from [benchmax::Tool](#).

**0.15.509.3.6 `getCommandline()` [1/2]** `virtual std::string benchmax::Tool::getCommandline ( const std::string & file ) const [inline], [virtual], [inherited]`  
 Compose commandline for this tool and the given input file.

**0.15.509.3.7 `getCommandline()` [2/2]** `virtual std::string benchmax::Tool::getCommandline ( const std::string & file, const std::string & localBinary ) const [inline], [virtual], [inherited]`  
 Compose commandline for this tool with another binary name and the given input file.

**0.15.509.3.8 `name()`** `std::string benchmax::Tool::name ( ) const [inline], [inherited]`  
 Common name of this tool.

**0.15.509.3.9 `parseCommandLine()`** `virtual std::optional<std::string> benchmax::Tool::parseCommandline ( const std::string & cmdline ) const [inline], [virtual], [inherited]`  
 Compose commandline for this tool and the given input file.

**0.15.509.3.10 `resolveDependencies()`** `std::vector<std::string> benchmax::Tool::resolveDependencies ( ) const [inline], [inherited]`  
 Get dependencies of binary required to run it (via ldd)

#### 0.15.509.4 Field Documentation

**0.15.509.4.1 `mArguments`** `std::string benchmax::Tool::mArguments [protected], [inherited]`  
 Command line arguments that should be passed to the binary.

**0.15.509.4.2 `mAttributes`** `std::map<std::string, std::string> benchmax::Tool::mAttributes [protected], [inherited]`  
 Attributes of the tool obtained by introspection of the binary.

**0.15.509.4.3 `mBinary`** `fs::path benchmax::Tool::mBinary [protected], [inherited]`  
 Path to the binary.

**0.15.509.4.4 `mName`** `std::string benchmax::Tool::mName [protected], [inherited]`  
 (Non-unique) identifier for the tool.

### 0.15.510 `benchmax::SMTRAT_OPB` Class Reference

Adapts the [SMTRAT](#) solver for OPB files.

```
#include <SMTRAT_OPB.h>
```

#### Public Member Functions

- **`SMTRAT_OPB`** (`const fs::path &binary, const std::string &arguments`)
   
*Adds "-opb" to instruct SMT-RAT to parse .opb files.*
- **`virtual bool canHandle`** (`const fs::path &path`) `const override`
  
*Only handles .opb files.*

- std::string [getStatusFromOutput](#) (const [BenchmarkResult](#) &result) const  
*Try to parse memouts from stderr.*
- bool [parse\\_stats](#) ([BenchmarkResult](#) &result) const
- virtual void [additionalResults](#) (const fs::path &, [BenchmarkResult](#) &result) const override  
*Computes the answer string from the exit code and parses statistics output.*
- std::string [name](#) () const  
*Common name of this tool.*
- fs::path [binary](#) () const  
*Full path to the binary.*
- const std::map< std::string, std::string > & [attributes](#) () const  
*A set of attributes, for example compilation options.*
- std::vector< std::string > [resolveDependencies](#) () const  
*Get dependencies of binary required to run it (via ldd)*
- std::size\_t [attributeHash](#) () const  
*Hash of the attributes.*
- virtual std::string [getCommandline](#) (const std::string &file) const  
*Compose commandline for this tool and the given input file.*
- virtual std::string [getCommandline](#) (const std::string &file, const std::string &localBinary) const  
*Compose commandline for this tool with another binary name and the given input file.*
- virtual std::optional< std::string > [parseCommandline](#) (const std::string &cmdline) const  
*Compose commandline for this tool and the given input file.*

## Protected Attributes

- std::string [mName](#)  
*(Non-unique) identifier for the tool.*
- fs::path [mBinary](#)  
*Path to the binary.*
- std::string [mArguments](#)  
*Command line arguments that should be passed to the binary.*
- std::map< std::string, std::string > [mAttributes](#)  
*Attributes of the tool obtained by introspection of the binary.*

### 0.15.510.1 Detailed Description

Adapts the [SMTRAT](#) solver for OPB files.

### 0.15.510.2 Constructor & Destructor Documentation

**0.15.510.2.1 [SMTRAT\\_OPB\(\)](#)** `benchmax::SMTRAT_OPB::SMTRAT_OPB (`  
  `const fs::path & binary,`  
  `const std::string & arguments ) [inline]`

Adds "-opb" to instruct SMT-RAT to parse .opb files.

### 0.15.510.3 Member Function Documentation

**0.15.510.3.1 [additionalResults\(\)](#)** `virtual void benchmax::SMTRAT::additionalResults (`  
  `const fs::path &,`  
  `BenchmarkResult & result ) const [inline], [override], [virtual], [inherited]`

Computes the answer string from the exit code and parses statistics output.

Reimplemented from [benchmax::Tool](#).

**0.15.510.3.2 attributeHash()** `std::size_t benchmax::Tool::attributeHash () const [inline], [inherited]`  
Hash of the attributes.

**0.15.510.3.3 attributes()** `const std::map<std::string, std::string>& benchmax::Tool::attributes () const [inline], [inherited]`  
A set of attributes, for example compilation options.

**0.15.510.3.4 binary()** `fs::path benchmax::Tool::binary () const [inline], [inherited]`  
Full path to the binary.

**0.15.510.3.5 canHandle()** `virtual bool benchmax::SMTRAT_OPB::canHandle (const fs::path & path) const [inline], [override], [virtual]`  
Only handles .opb files.  
Reimplemented from `benchmax::SMTRAT`.

**0.15.510.3.6 getCommandLine() [1/2]** `virtual std::string benchmax::Tool::getCommandLine (const std::string & file) const [inline], [virtual], [inherited]`  
Compose commandline for this tool and the given input file.

**0.15.510.3.7 getCommandLine() [2/2]** `virtual std::string benchmax::Tool::getCommandLine (const std::string & file, const std::string & localBinary) const [inline], [virtual], [inherited]`  
Compose commandline for this tool with another binary name and the given input file.

**0.15.510.3.8 getStatusFromOutput()** `std::string benchmax::SMTRAT::getStatusFromOutput (const BenchmarkResult & result) const [inline], [inherited]`  
Try to parse memouts from stderr.

**0.15.510.3.9 name()** `std::string benchmax::Tool::name () const [inline], [inherited]`  
Common name of this tool.

**0.15.510.3.10 parse\_stats()** `bool benchmax::SMTRAT::parse_stats (BenchmarkResult & result) const [inline], [inherited]`

**0.15.510.3.11 parseCommandLine()** `virtual std::optional<std::string> benchmax::Tool::parseCommandline (const std::string & cmdline) const [inline], [virtual], [inherited]`  
Compose commandline for this tool and the given input file.

**0.15.510.3.12 resolveDependencies()** `std::vector<std::string> benchmax::Tool::resolveDependencies () const [inline], [inherited]`  
Get dependencies of binary required to run it (via ldd)

#### 0.15.510.4 Field Documentation

**0.15.510.4.1 mArguments** std::string benchmax::Tool::mArguments [protected], [inherited]  
Command line arguments that should be passed to the binary.

**0.15.510.4.2 mAttributes** std::map<std::string, std::string> benchmax::Tool::mAttributes [protected], [inherited]  
Attributes of the tool obtained by introspection of the binary.

**0.15.510.4.3 mBinary** fs::path benchmax::Tool::mBinary [protected], [inherited]  
Path to the binary.

**0.15.510.4.4 mName** std::string benchmax::Tool::mName [protected], [inherited]  
(Non-unique) identifier for the tool.

### 0.15.511 smtrat::execution::SoftAssertion Struct Reference

```
#include <ExecutionState.h>
```

#### Data Fields

- FormulaT formula
- Rational weight
- std::string id

#### 0.15.511.1 Field Documentation

**0.15.511.1.1 formula** FormulaT smtrat::execution::SoftAssertion::formula

**0.15.511.1.2 id** std::string smtrat::execution::SoftAssertion::id

**0.15.511.1.3 weight** Rational smtrat::execution::SoftAssertion::weight

### 0.15.512 smtrat::settings::SolverSettings Struct Reference

```
#include <Settings.h>
```

#### Data Fields

- bool print\_model
- bool print\_all\_models
- bool preprocess
- std::string preprocess\_output\_file
- bool convert\_to\_cnf\_dimacs
- bool convert\_to\_cnf\_smtlib

### 0.15.512.1 Field Documentation

**0.15.512.1.1 convert\_to\_cnf\_dimacs** bool smtrat::settings::SolverSettings::convert\_to\_cnf\_↔ dimacs

**0.15.512.1.2 convert\_to\_cnf\_smtlib** bool smtrat::settings::SolverSettings::convert\_to\_cnf\_smtlib

**0.15.512.1.3 preprocess** bool smtrat::settings::SolverSettings::preprocess

**0.15.512.1.4 preprocess\_output\_file** std::string smtrat::settings::SolverSettings::preprocess\_↔ output\_file

**0.15.512.1.5 print\_all\_models** bool smtrat::settings::SolverSettings::print\_all\_models

**0.15.512.1.6 print\_model** bool smtrat::settings::SolverSettings::print\_model

## 0.15.513 smtrat::parser::SortedVariableParser Struct Reference

```
#include <Term.h>
```

### Public Member Functions

- [SortedVariableParser \(\)](#)

### Data Fields

- [SymbolParser symbol](#)
- [SortParser sort](#)
- [qi::rule<Iterator, std::pair<std::string, carl::Sort>, Skipper> main](#)

### 0.15.513.1 Constructor & Destructor Documentation

**0.15.513.1.1 SortedVariableParser()** smtrat::parser::SortedVariableParser::SortedVariableParser ( ) [inline]

### 0.15.513.2 Field Documentation

**0.15.513.2.1 main** qi::rule<[Iterator](#), std::pair<std::string, carl::Sort>, [Skipper](#)> smtrat↔ ::parser::SortedVariableParser::main

**0.15.513.2.2 sort** SortParser smtrat::parser::SortedVariableParser::sort

**0.15.513.2.3 symbol** SymbolParser smtrat::parser::SortedVariableParser::symbol

## 0.15.514 smtrat::parser::SortParser Struct Reference

```
#include <Sort.h>
```

### Public Member Functions

- `SortParser ()`
- `void setParameters (const std::vector< std::string > &params)`
- `void clearParameters ()`
- `carl::Sort getSort (const Identifier &i)`
- `carl::Sort getSortWithParam (const Identifier &i, const std::vector< carl::Sort > &params)`

### 0.15.514.1 Constructor & Destructor Documentation

**0.15.514.1.1 SortParser()** smtrat::parser::SortParser::SortParser ( ) [inline]

### 0.15.514.2 Member Function Documentation

**0.15.514.2.1 clearParameters()** void smtrat::parser::SortParser::clearParameters ( ) [inline]

**0.15.514.2.2 getSort()** carl::Sort smtrat::parser::SortParser::getSort ( const Identifier & i ) [inline]

**0.15.514.2.3 getSortWithParam()** carl::Sort smtrat::parser::SortParser::getSortWithParam ( const Identifier & i, const std::vector< carl::Sort > & params ) [inline]

**0.15.514.2.4 setParameters()** void smtrat::parser::SortParser::setParameters ( const std::vector< std::string > & params ) [inline]

## 0.15.515 smtrat::parser::SpecConstantParser Struct Reference

```
#include <SExpression.h>
```

### Public Member Functions

- `SpecConstantParser ()`

### Data Fields

- `NumeralParser numeral`
- `DecimalParser decimal`
- `HexadecimalParser hexadecimal`
- `BinaryParser binary`
- `StringParser string`
- `qi::symbols< char, types::ConstType > theoryConst`
- `qi::rule< Iterator, types::ConstType(), Skipper > main`

### 0.15.515.1 Constructor & Destructor Documentation

**0.15.515.1.1 SpecConstantParser()** smtrat::parser::SpecConstantParser::SpecConstantParser ( )  
[inline]

## 0.15.515.2 Field Documentation

**0.15.515.2.1 binary** BinaryParser smtrat::parser::SpecConstantParser::binary

**0.15.515.2.2 decimal** DecimalParser smtrat::parser::SpecConstantParser::decimal

**0.15.515.2.3 hexadecimal** HexadecimalParser smtrat::parser::SpecConstantParser::hexadecimal

**0.15.515.2.4 main** qi::rule<Iterator, types::ConstType(), Skipper> smtrat::parser::SpecConstantParser::main

**0.15.515.2.5 numeral** NumeralParser smtrat::parser::SpecConstantParser::numeral

**0.15.515.2.6 string** StringParser smtrat::parser::SpecConstantParser::string

**0.15.515.2.7 theoryConst** qi::symbols<char, types::ConstType> smtrat::parser::SpecConstantParser::theoryConst

## 0.15.516 smtrat::SplitSOSModule< Settings > Class Template Reference

#include <SplitSOSModule.h>

### Public Types

- typedef Settings SettingsType
- enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }

### Public Member Functions

- std::string moduleName () const
- SplitSOSModule (const ModuleInput \*\_formula, Conditionals &\_conditionals, Manager \*\_manager=NULL)
- ~SplitSOSModule ()
- Answer checkCore ()
 

*Checks the received formula for consistency.*
- bool isPreprocessor () const
- bool appliedPreprocessing () const
- bool add (ModuleInput::const\_iterator \_subformula)
 

*The module has to take the given sub-formula of the received formula into account.*
- void remove (ModuleInput::const\_iterator \_subformula)
 

*Removes everything related to the given sub-formula of the received formula.*
- Answer check (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_VARIABLE)
 

*Checks the received formula for consistency.*
- Answer runBackends (bool \_final, bool \_full, carl::Variable \_objective)
 

*Runs the backend solvers on the passed formula.*

- virtual `Answer runBackends ()`
- std::pair< bool, `FormulaT` > `getReceivedFormulaSimplified ()`
- void `updateModel () const`

*Updates the current assignment into the model.*
- bool `inform (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- void `deinform (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- virtual void `init ()`

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- virtual void `updateAllModels ()`

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > `getModelEqualities () const`

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned `currentlySatisfiedByBackend (const FormulaT &_formula) const`
- virtual unsigned `currentlySatisfied (const FormulaT &) const`
- bool `receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- std::size\_t `id () const`
- void `setId (std::size_t _id)`

*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- void `setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput * pReceivedFormula () const`
- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const std::vector< `Model` > & `allModels () const`
- const std::vector< `FormulaSetT` > & `infeasibleSubsets () const`
- const std::vector< `Module` \* > & `usedBackends () const`
- const carl::FastSet< `FormulaT` > & `constraintsToInform () const`
- const carl::FastSet< `FormulaT` > & `informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=Lemmatype::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormalType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const std::vector< `Lemma` > & `lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

- Stores all lemmas of any backend of this module in its own lemma vector.
- void `collectTheoryPropagations ()`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- void `print (const std::string &_initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string &_initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string &_initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string &_initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream &_out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream &_out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore = 5`

*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`

*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches = 0`

*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`

*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- virtual void `deinformCore (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- virtual bool `addCore (ModuleInput::const_iterator formula)`

*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore (ModuleInput::const_iterator formula)`

*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound () const`

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*

- void `clearModel () const`  
*Clears the assignment, if any was found.*
- void `clearModels () const`  
*Clears all assignments, if any was found.*
- void `cleanModel () const`  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin ()`
- `ModuleInput::iterator passedFormulaEnd ()`
- void `addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
- void `getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
- void `informBackends (const FormulaT &_constraint)`  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform (const FormulaT &_constraint)`  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula (const FormulaT &_origin) const`
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula (const FormulaT &_formula)`  
*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`  
*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`  
*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset ()`  
*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`  
*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin (const std::vector< FormulaT > &_origins) const`
- void `getInfeasibleSubsets ()`  
*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets (const Module &_backend) const`  
*Get the infeasible subsets the given backend provides.*
- const `Model & backendsModel () const`  
*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel () const`
- void `getBackendsAllModels () const`  
*Stores all models of a backend in the list of all models of this module.*
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`  
*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`

- std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const  
*Merges the two vectors of sets into the first one.*
- size\_t [determine\\_smallest\\_origin](#) (const std::vector< [FormulaT](#) > &origins) const
- bool [probablyLooping](#) (const typename Poly::PolyType &\_branchingPolynomial, const Rational &← branchingValue) const  
*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool [branchAt](#) (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)  
*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &← \_premise=std::vector< [FormulaT](#) >())
- bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool [branchAt](#) (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
- void [splitUnequalConstraint](#) (const [FormulaT](#) &)  
*Adds the following lemmas for the given constraint p!=0.*
- unsigned [checkModel](#) () const
- void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()  
*Gets all InformationRelevantFormulas.*
- bool [isLemmaLevel](#) ([LemmaLevel](#) level)  
*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool [modelsDisjoint](#) (const Model &\_modelA, const Model &\_modelB)  
*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< [FormulaSetT](#) > [mInfeasibleSubsets](#)  
*Stores the infeasible subsets.*
- Manager \*const [mpManager](#)  
*A reference to the manager.*
- Model [mModel](#)  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< Model > [mAllModels](#)  
*Stores all satisfying assignments.*
- bool [mModelComputed](#)  
*True, if the model has already been computed.*
- bool [mFinalCheck](#)  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool [mFullCheck](#)  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable [mObjectiveVariable](#)  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< TheoryPropagation > [mTheoryPropagations](#)

- std::atomic< [Answer](#) > mSolverState
  - States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* mBackendsFoundAnswer
  - This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- [Conditionals](#) mFoundAnswer
  - Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< [Module](#) \* > mUsedBackends
  - The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< [Module](#) \* > mAllBackends
  - The backends of this module which have been used.*
- std::vector< [Lemma](#) > mLemmas
  - Stores the lemmas being valid formulas this module or its backends made.*
- [ModuleInput](#)::iterator mFirstSubformulaToPass
  - Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< [FormulaT](#) > mConstraintsToInform
  - Stores the constraints which the backends must be informed about.*
- carl::FastSet< [FormulaT](#) > mInformedConstraints
  - Stores the position of the first constraint of which no backend has been informed about.*
- [ModuleInput](#)::const\_iterator mFirstUncheckedReceivedSubformula
  - Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned mSmallerMusesCheckCounter
  - Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > mVariableCounters
  - Maps variables to the number of their occurrences.*

### 0.15.516.1 Member Typedef Documentation

**0.15.516.1.1 SettingsType** template<typename Settings >  
typedef [smtrat::SplitsOSModule](#)< Settings >::SettingsType

### 0.15.516.2 Member Enumeration Documentation

**0.15.516.2.1 LemmaType** enum [smtrat::Module](#)::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.516.3 Constructor & Destructor Documentation

**0.15.516.3.1 SplitSOSModule()** template<typename Settings >  
[smtrat::SplitSOSModule](#)< Settings >::SplitSOSModule (  
    const [ModuleInput](#) \* \_formula,

```
Conditionals & _conditionals,
Manager * _manager = NULL)
```

### 0.15.516.3.2 ~SplitSOSModule() template<typename Settings > smrat::SplitSOSModule< Settings >::~SplitSOSModule ( )

#### 0.15.516.4 Member Function Documentation

##### 0.15.516.4.1 add() bool smrat::PModule::add (

`ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

###### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

###### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

##### 0.15.516.4.2 addConstraintToInform() void smrat::Module::addConstraintToInform (

`const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

###### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in `smrat::SATModule< Settings >`.

##### 0.15.516.4.3 addCore() virtual bool smrat::Module::addCore (

`ModuleInput::const_iterator formula ) [inline], [protected], [virtual], [inherited]`

The module has to take the given sub-formula of the received formula into account.

###### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

###### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in `smrat::VSModule< Settings >`, `smrat::SATModule< Settings >`, `smrat::PFEModule< Settings >`, `smrat::ICPModule< Settings >`, `smrat::ICPModule< ICPSettings4 >`, `smrat::ICPModule< ICPSettings1 >`, `smrat::UnionFindModule< Settings >`, `smrat::UFCegarModule< Settings >`, `smrat::STropModule< Settings >`, `smrat::PBPPModule< Settings >`, `smrat::PBGaussModule< Settings >`, `smrat::NRAILModule< Settings >`, `smrat::NewCoveringModule< Settings >`, `smrat::NewCADModule< Settings >`, `smrat::LRAModule< Settings >`, `smrat::LRAModule< LRASettingsICP >`, `smrat::LRAModule< LRASettings1 >`, `smrat::IntEqModule< Settings >`, `smrat::IntBlastModule< Settings >`, `smrat::IncWidthModule< Settings >`, `smrat::ICEModule< Settings >`,

`smrat::FPPModule< Settings >, smrat::FouMoModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::CubeLIAModule< Settings >, smrat::CSplitModule< Settings >, smrat::BVMModule< Settings >, and smrat::GBModule< Settings >.`

**0.15.516.4.4 addInformationRelevantFormula()** `void smrat::Module::addInformationRelevantFormula(`

`const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.516.4.5 addLemma()** `void smrat::Module::addLemma(`

`const FormulaT & _lemma,`

`const LemmaType & _lt = LemmaType::NORMAL,`

`const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline],`

[inherited]

Stores a lemma being a valid formula.

#### Parameters

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.516.4.6 addOrigin()** `void smrat::Module::addOrigin(`

`ModuleInput::iterator _formula,`

`const FormulaT & _origin ) [inline], [protected], [inherited]`

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.516.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator, bool> smrat::Module::addReceivedSubformulaToPassedFormula(`

`ModuleInput::const_iterator _subformula ) [inline], [protected], [inherited]`

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

```
0.15.516.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat<->
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.516.4.9 addSubformulaToPassedFormula() [2/3] std::pair<ModuleInput::iterator,bool> smtrat<->
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.516.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
```

```
smtrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.516.4.11 allModels()** const std::vector<[Model](#)>& smtrat::Module::allModels () const [inline], [inherited]

**Returns**

All satisfying assignments, if existent.

**0.15.516.4.12 anAnswerFound()** bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.516.4.13 answerFound()** const [smtrat::Conditionals](#)& smtrat::Module::answerFound () const [inline], [inherited]

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.516.4.14 appliedPreprocessing()** bool smtrat::PModule::appliedPreprocessing () const [inline], [inherited]

**Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.516.4.15 backendsModel()** const [Model](#) & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.516.4.16 branchAt() [1/4]** bool smtrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector< FormulaT > ()) [inline],
[protected], [inherited]
```

**0.15.516.4.17 branchAt() [2/4]** bool smtrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

```
0.15.516.4.18 branchAt() [3/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

```
0.15.516.4.19 branchAt() [4/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.516.4.20 check() Answer smtrat::PModule::check (
```

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.516.4.21 checkCore()** `template<typename Settings >  
Answer smrat::SplitsSOSModule< Settings >::checkCore () [virtual]  
Checks the received formula for consistency.`

**Returns**

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.516.4.22 checkInfSubsetForMinimality()** `void smrat::Module::checkInfSubsetForMinimality (  
std::vector< FormulaSetT >::const_iterator _infsSubset,  
const std::string & _filename = "smaller_muses",  
unsigned _maxSizeDifference = 1 ) const [inherited]`

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.516.4.23 checkModel()** `unsigned smrat::Module::checkModel () const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.516.4.24 cleanModel()** `void smrat::Module::cleanModel () const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.516.4.25 clearLemmas()** `void smrat::Module::clearLemmas () [inline], [inherited]`

Deletes all yet found lemmas.

**0.15.516.4.26 clearModel()** `void smrat::Module::clearModel () const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.516.4.27 `clearModels()`** `void smtrat::Module::clearModels () const [inline], [protected], [inherited]`  
 Clears all assignments, if any was found.

**0.15.516.4.28 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula () [protected], [inherited]`

**0.15.516.4.29 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins (const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.516.4.30 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins (const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.516.4.31 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations () [inherited]`

**0.15.516.4.32 `constraintsToInform()`** `const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]`

#### Returns

The constraints which the backends must be informed about.

**0.15.516.4.33 `currentlySatisfied()`** `virtual unsigned smtrat::Module::currentlySatisfied (const FormulaT & ) const [inline], [virtual], [inherited]`

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in `smtrat::LRAModule< Settings >`, `smtrat::LRAModule< LRASettingsICP >`, and `smtrat::LRAModule< LRASettings >`

**0.15.516.4.34 `currentlySatisfiedByBackend()`** `unsigned smtrat::Module::currentlySatisfiedByBackend (const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.516.4.35 `deinform()`** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.516.4.36 `deinformCore()`** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAbstractModule< Settings >](#), [smtrat::LRAbstractModule< LRASettingsICP >](#), and [smtrat::LRAbstractModule< LRASettings >](#)

**0.15.516.4.37 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of `origins`

**0.15.516.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::erasesubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.516.4.39 excludeNotReceivedVariablesFromModel()** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`  
 Excludes all variables from the current model, which do not occur in the received formula.

**0.15.516.4.40 findBestOrigin()** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`  
`const std::vector< FormulaT > & _origins ) const [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.516.4.41 firstSubformulaToPass()** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.516.4.42 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.516.4.43 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable (`  
`const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.516.4.44 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.516.4.45 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.516.4.46 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.516.4.47 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]

Copies the infeasible subsets of the passed formula.

**0.15.516.4.48 getInfeasibleSubsets() [2/2]** std::vector< [FormulaSetT](#) > smtrat::Module::getInfeasibleSubsets (

const [Module](#) & *\_backend* ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>_backend</i> | The backend from which to obtain the infeasible subsets. |
|-----------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.516.4.49 getInformationRelevantFormulas()** const std::set< [FormulaT](#) > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]

Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.516.4.50 getModelEqualities()** std::list< std::vector< [carl::Variable](#) > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.516.4.51 getOrigins()** [1/3] void smtrat::Module::getOrigins (

const [FormulaT](#) & *\_formula*,

[FormulaSetT](#) & *\_origins* ) const [inline], [protected], [inherited]

#### Parameters

|                 |  |
|-----------------|--|
| <i>_formula</i> |  |
| <i>_origins</i> |  |

**0.15.516.4.52 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

#### Parameters

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.516.4.53 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

#### Parameters

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.516.4.54 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.516.4.55 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.516.4.56 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

#### Returns

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.516.4.57 id()** std::size\_t smtrat::Module::id ( ) const [inline], [inherited]

#### Returns

A unique ID to identify this module instance.

**0.15.516.4.58 infeasibleSubsets()** const std::vector<[FormulaSetT](#)>& smrat::Module::infeasibleSubsets () const [inline], [inherited]

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.516.4.59 inform()** bool smrat::Module::inform (const [FormulaT](#) & \_constraint ) [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.516.4.60 informBackends()** void smrat::Module::informBackends (const [FormulaT](#) & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.516.4.61 informCore()** virtual bool smrat::Module::informCore (const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.516.4.62 informedConstraints()** const carl::FastSet<[FormulaT](#)>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]

#### Returns

The position of the first constraint of which no backend has been informed about.

**0.15.516.4.63 init()** void smtrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), and [smtrat::BVModule< Settings >](#).

**0.15.516.4.64 is\_minimizing()** bool smtrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.516.4.65 isLemmaLevel()** bool smtrat::Module::isLemmaLevel (

[LemmaLevel](#) level ) [protected], [inherited]

Checks if current lemma level is greater or equal to given level.

#### Parameters

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.516.4.66 isPreprocessor()** bool smtrat::PModule::isPreprocessor ( ) const [inline], [inherited]

#### Returns

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.516.4.67 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

#### Returns

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.516.4.68 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge (

const std::vector< [FormulaT](#) > & \_vecSetA,

const std::vector< [FormulaT](#) > & \_vecSetB ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

#### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.516.4.69 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.516.4.70 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (`

`const Model & _modelA,`

`const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.516.4.71 `moduleName()`** `template<typename Settings >`

`std::string smtrat::SplitsOSModule< Settings >::moduleName () const [inline], [virtual]`

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.516.4.72 `objective()`** `carl::Variable smtrat::Module::objective () const [inline], [inherited]`**0.15.516.4.73 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`

`const FormulaT & _origin ) const [protected], [inherited]`

**0.15.516.4.74 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin () [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.516.4.75 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd () [inline], [protected], [inherited]`**Returns**

An iterator to the end of the passed formula.

**0.15.516.4.76 pPassedFormula()** const `ModuleInput*` smtrat::Module::pPassedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.516.4.77 pReceivedFormula()** const `ModuleInput*` smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.516.4.78 print()** void smtrat::Module::print (const std::string & `_initiation` = "\*\*\*") const [inherited]

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.516.4.79 printAllModels()** void smtrat::Module::printAllModels (std::ostream & `_out` = `std::cout`) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.516.4.80 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets (const std::string & `_initiation` = "\*\*\*") const [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.516.4.81 printModel()** void smtrat::Module::printModel (std::ostream & `_out` = `std::cout`) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.516.4.82 printPassedFormula()** void smrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the vector of passed formula.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.516.4.83 printReceivedFormula()** void smrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the vector of the received formula.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.516.4.84 probablyLooping()** bool smrat::Module::probablyLooping ( const typename Poly::PolyType & *\_branchingPolynomial*, const Rational & *\_branchingValue* ) const [protected], [inherited]  
Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.516.4.85 receivedFormulaChecked()** void smrat::Module::receivedFormulaChecked ( ) [inline], [inherited]  
Notifies that the received formulas has been checked.

**0.15.516.4.86 receivedFormulasAsInfeasibleSubset()** void smrat::Module::receivedFormulasAsInfeasibleSubset ( ModuleInput::const\_iterator *\_subformula* ) [inline], [protected], [inherited]  
Stores an infeasible subset consisting only of the given received formula.

**0.15.516.4.87 receivedVariable()** bool smrat::Module::receivedVariable ( carl::Variable::Arg *\_var* ) const [inline], [inherited]

**0.15.516.4.88 remove()** void smrat::PModule::remove ( ModuleInput::const\_iterator *\_subformula* ) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.516.4.89 `removeCore()`** `virtual void smtrat::Module::removeCore ( ModuleInput::const_iterator formula ) [inline], [protected], [virtual], [inherited]`

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICEModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::GBModule< Settings >](#).

**0.15.516.4.90 `removeOrigin()`** `std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin ( ModuleInput::iterator _formula, const FormulaT & _origin ) [inline], [protected], [inherited]`

**0.15.516.4.91 `removeOrigins()`** `std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins ( ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected], [inherited]`

**0.15.516.4.92 `rPassedFormula()`** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.516.4.93 `rReceivedFormula()`** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A reference to the formula passed to this module.

**0.15.516.4.94 runBackends() [1/2]** `virtual Answer smtrat::PModule::runBackends () [inline], [virtual], [inherited]`

Reimplemented from [smtrat::Module](#).

**0.15.516.4.95 runBackends() [2/2]** `Answer smtrat::PModule::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.516.4.96 setId()** `void smtrat::Module::setId (`

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

**Parameters**

|                   |                                    |
|-------------------|------------------------------------|
| <code>↔ id</code> | The id to set this module's id to. |
|-------------------|------------------------------------|

**0.15.516.4.97 setThreadPriority()** `void smtrat::Module::setThreadPriority (`

```
 thread_priority _threadPriority) [inline], [inherited]
```

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.516.4.98 solverState()** `Answer smtrat::Module::solverState () const [inline], [inherited]`

**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.516.4.99 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint (`

`const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.516.4.100 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.516.4.101 updateAllModels()** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.516.4.102 updateLemmas()** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.516.4.103 updateModel()** `void smtrat::PModule::updateModel ( ) const [virtual], [inherited]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.516.4.104 usedBackends()** `const std::vector<Module*>& smtrat::Module::usedBackends ( )`

`const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.516.5 Field Documentation****0.15.516.5.1 mAllBackends** `std::vector<Module*> smtrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.516.5.2 mAllModels** `std::vector<Model>` `smtrat::Module::mAllModels` [mutable], [protected], [inherited]  
Stores all satisfying assignments.

**0.15.516.5.3 mBackendsFoundAnswer** `std::atomic_bool*` `smtrat::Module::mBackendsFoundAnswer` [protected], [inherited]  
This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.516.5.4 mConstraintsToInform** `carl::FastSet<FormulaT>` `smtrat::Module::mConstraintsToInform` [protected], [inherited]  
Stores the constraints which the backends must be informed about.

**0.15.516.5.5 mFinalCheck** `bool` `smtrat::Module::mFinalCheck` [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.516.5.6 mFirstPosInLastBranches** `std::size_t` `smtrat::Module::mFirstPosInLastBranches` = 0 [static], [inherited]  
The beginning of the cyclic buffer storing the last branches.

**0.15.516.5.7 mFirstSubformulaToPass** `ModuleInput::iterator` `smtrat::Module::mFirstSubformulaToPass` [protected], [inherited]  
Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.516.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator` `smtrat::Module::mFirstUncheckedReceivedSubformula` [protected], [inherited]  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.516.5.9 mFoundAnswer** `Conditionals` `smtrat::Module::mFoundAnswer` [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.516.5.10 mFullCheck** `bool` `smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.516.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT>` `smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.516.5.12 mInformedConstraints** `carl::FastSet<FormulaT>` `smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.516.5.13 mLastBranches** std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]

Stores the last branches in a cycle buffer.

**0.15.516.5.14 mLemmas** std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]

Stores the lemmas being valid formulas this module or its backends made.

**0.15.516.5.15 mModel** Model smtrat::Module::mModel [mutable], [protected], [inherited]

Stores the assignment of the current satisfiable result, if existent.

**0.15.516.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]

True, if the model has already been computed.

**0.15.516.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore =

5 [static], [inherited]

The number of different variables to consider for a probable infinite loop of branchings.

**0.15.516.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]

Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.516.5.19 mOldSplittingVariables** std::vector< FormulaT > smtrat::Module::mOldSplittingVariables [static], [inherited]

Reusable splitting variables.

**0.15.516.5.20 mpManager** Manager\* const smtrat::Module::mpManager [protected], [inherited]

A reference to the manager.

**0.15.516.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]

Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.516.5.22 mSolverState** std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.516.5.23 mTheoryPropagations** std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]

**0.15.516.5.24 mUsedBackends** std::vector<[Module](#)\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.516.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.517 benchmax::SSHBackend Class Reference

```
#include <SSHBackend.h>
```

### Public Member Functions

- [SSHBackend \(\)](#)
- [~SSHBackend \(\)](#)
- void [run \(const Jobs &, bool\)](#)  
*Dummy if SSH is disabled.*
- bool [suspendable \(\) const](#)
- void [process\\_results \(const Jobs &jobs, bool check\\_finished\)](#)
- void [addResult \(const Tool \\*tool, const fs::path &file, BenchmarkResult &&result\)](#)  
*Add a result.*

### Protected Member Functions

- virtual void [startTool \(const Tool \\*\)](#)  
*Hook for every tool at the beginning.*
- virtual void [finalize \(\)](#)  
*Hook to allow for asynchronous backends to wait for jobs to terminate.*
- virtual void [execute \(const Tool \\*, const fs::path &\)](#)  
*Execute a single pair of tool and benchmark.*
- void [madeProgress \(std::size\\_t files=1\)](#)  
*Can be called to give information about the current progress, if available.*
- virtual bool [collect\\_results \(const Jobs &, bool\)](#)
- void [sanitize\\_results \(const Jobs &jobs\) const](#)
- void [write\\_results \(const Jobs &jobs\) const](#)

### Protected Attributes

- std::size\_t [mExpectedJobs](#)  
*Number of jobs that should be run.*
- std::atomic< std::size\_t > [mFinishedJobs](#)  
*Number of jobs that are finished.*
- std::atomic< std::size\_t > [mLastPercent](#)  
*Percentage of finished jobs when [madeProgress\(\)](#) was last called.*

### 0.15.517.1 Constructor & Destructor Documentation

#### 0.15.517.1.1 [SSHBackend\(\)](#) benchmax::SSHBackend::SSHBackend ( ) [inline]

**0.15.517.1.2 ~SSHBackend()** `benchmax::SSHBackend::~SSHBackend( ) [inline]`

### 0.15.517.2 Member Function Documentation

**0.15.517.2.1 addResult()** `void benchmax::Backend::addResult( const Tool * tool, const fs::path & file, BenchmarkResult && result ) [inline], [inherited]`

Add a result.

**0.15.517.2.2 collect\_results()** `virtual bool benchmax::Backend::collect_results( const Jobs & , bool ) [inline], [protected], [virtual], [inherited]`

**0.15.517.2.3 execute()** `virtual void benchmax::Backend::execute( const Tool * , const fs::path & ) [inline], [protected], [virtual], [inherited]`

Execute a single pair of tool and benchmark.

Reimplemented in [benchmax::LocalBackend](#), and [benchmax::CondorBackend](#).

**0.15.517.2.4 finalize()** `virtual void benchmax::Backend::finalize( ) [inline], [protected], [virtual], [inherited]`

Hook to allow for asynchronous backends to wait for jobs to terminate.

**0.15.517.2.5 madeProgress()** `void benchmax::Backend::madeProgress( std::size_t files = 1 ) [inline], [protected], [inherited]`

Can be called to give information about the current progress, if available.

**0.15.517.2.6 process\_results()** `void benchmax::Backend::process_results( const Jobs & jobs, bool check_finished ) [inline], [inherited]`

**0.15.517.2.7 run()** `void benchmax::SSHBackend::run( const Jobs & , bool ) [inline]`

Dummy if SSH is disabled.

**0.15.517.2.8 sanitize\_results()** `void benchmax::Backend::sanitize_results( const Jobs & jobs ) const [inline], [protected], [inherited]`

**0.15.517.2.9 startTool()** `virtual void benchmax::Backend::startTool( const Tool * ) [inline], [protected], [virtual], [inherited]`

Hook for every tool at the beginning.

Can be used to upload the tool to some remote system.

**0.15.517.2.10 suspendable()** `bool benchmax::Backend::suspendable () const [inline], [inherited]`

**0.15.517.2.11 write\_results()** `void benchmax::Backend::write_results (const Jobs & jobs) const [inline], [protected], [inherited]`

### 0.15.517.3 Field Documentation

**0.15.517.3.1 mExpectedJobs** `std::size_t benchmax::Backend::mExpectedJobs [protected], [inherited]`  
Number of jobs that should be run.

**0.15.517.3.2 mFinishedJobs** `std::atomic<std::size_t> benchmax::Backend::mFinishedJobs [protected], [inherited]`  
Number of jobs that are finished.

**0.15.517.3.3 mLastPercent** `std::atomic<std::size_t> benchmax::Backend::mLastPercent [protected], [inherited]`  
Percentage of finished jobs when [madeProgress\(\)](#) was last called.

## 0.15.518 `benchmax::settings::SSHBackendSettings` Struct Reference

[Settings](#) for SSH backend.

```
#include <SSHSettings.h>
```

### Data Fields

- `std::vector< std::string > nodes`  
*List of nodes to connect to.*
- `std::string basedir`  
*Base directory for solvers.*
- `std::string tmpdir`  
*Temporary directory for benchmarks and output files.*
- `bool use_wallclock`  
*Use wallclock timeouts instead of CPU time.*
- `bool resolve_deps`  
*Resolve and upload dependencies of binary.*

### 0.15.518.1 Detailed Description

[Settings](#) for SSH backend.

### 0.15.518.2 Field Documentation

**0.15.518.2.1 basedir** `std::string benchmax::settings::SSHBackendSettings::basedir`  
Base directory for solvers.

**0.15.518.2.2 nodes** `std::vector<std::string> benchmax::settings::SSHBackendSettings::nodes`  
List of nodes to connect to.

**0.15.518.2.3 resolve\_deps** bool benchmax::settings::SSHBackendSettings::resolve\_deps  
Resolve and upload dependencies of binary.

**0.15.518.2.4 tmpdir** std::string benchmax::settings::SSHBackendSettings::tmpdir  
Temporary directory for benchmarks and output files.

**0.15.518.2.5 use\_wallclock** bool benchmax::settings::SSHBackendSettings::use\_wallclock  
Use wallclock timeouts instead of CPU time.

## 0.15.519 benchmax::ssh::SSHConnection Class Reference

A wrapper class that manages a single SSH connection as specified in a [Node](#) object (with all its channels).

```
#include <SSHConnection.h>
```

### Public Member Functions

- **SSHConnection** (const [Node](#) &n)  
*Create a new connection for the given node.*
- **~SSHConnection** ()  
*Wait for all channels to terminate.*
- const [Node](#) & **getNode** () const  
*Return the node.*
- bool **jobFree** ()  
*Check if a new job could be started.*
- void **newJob** ()  
*Increase job counter.*
- void **finishJob** ()  
*Decrease job counter.*
- std::size\_t **jobs** () const  
*Current number of jobs.*
- bool **busy** ()  
*Check if all channels are busy.*
- std::string **createTmpDir** (const std::string &folder)  
*Create a temporary directory on the remote.*
- void **removeDir** (const std::string &folder)  
*Remove a (temporary) directory on the remote.*
- bool **uploadFile** (const fs::path &local, const std::string &base, const std::string &remote, int mode=S\_←IRUSR|S\_IWUSR)  
*Upload a file to the remote.*
- bool **executeCommand** (const std::string &cmd, [BenchmarkResult](#) &result)  
*Execute a command on the remote.*

### 0.15.519.1 Detailed Description

A wrapper class that manages a single SSH connection as specified in a [Node](#) object (with all its channels).

### 0.15.519.2 Constructor & Destructor Documentation

**0.15.519.2.1 `SSHConnection()`** `benchmax::ssh::SSHConnection::SSHConnection (` `const Node & n ) [inline]`

Create a new connection for the given node.

**0.15.519.2.2 `~SSHConnection()`** `benchmax::ssh::SSHConnection::~SSHConnection ( ) [inline]`

Wait for all channels to terminate.

### 0.15.519.3 Member Function Documentation

**0.15.519.3.1 `busy()`** `bool benchmax::ssh::SSHConnection::busy ( ) [inline]`

Check if all channels are busy.

**0.15.519.3.2 `createTmpDir()`** `std::string benchmax::ssh::SSHConnection::createTmpDir (` `const std::string & folder ) [inline]`

Create a temporary directory on the remote.

**0.15.519.3.3 `executeCommand()`** `bool benchmax::ssh::SSHConnection::executeCommand (` `const std::string & cmd,` `BenchmarkResult & result ) [inline]`

Execute a command on the remote.

**0.15.519.3.4 `finishJob()`** `void benchmax::ssh::SSHConnection::finishJob ( ) [inline]`

Decrease job counter.

**0.15.519.3.5 `getNode()`** `const Node& benchmax::ssh::SSHConnection::getNode ( ) const [inline]`

Return the node.

**0.15.519.3.6 `jobFree()`** `bool benchmax::ssh::SSHConnection::jobFree ( ) [inline]`

Check if a new job could be started.

**0.15.519.3.7 `jobs()`** `std::size_t benchmax::ssh::SSHConnection::jobs ( ) const [inline]`

Current number of jobs.

**0.15.519.3.8 `newJob()`** `void benchmax::ssh::SSHConnection::newJob ( ) [inline]`

Increase job counter.

**0.15.519.3.9 `removeDir()`** `void benchmax::ssh::SSHConnection::removeDir (` `const std::string & folder ) [inline]`

Remove a (temporary) directory on the remote.

```
0.15.519.3.10 uploadFile() bool benchmax::ssh::SSHConnection::uploadFile (
 const fs::path & local,
 const std::string & base,
 const std::string & remote,
 int mode = S_IRUSR | S_IWUSR) [inline]
```

Upload a file to the remote.

## 0.15.520 smrat::vs::State Class Reference

```
#include <State.h>
```

### Public Types

- enum `Type` { `TEST_CANDIDATE_TO_GENERATE` , `SUBSTITUTION_TO_APPLY` , `COMBINE_SUBRESULTS` }
- typedef `carl::FastPointerMap< Substitution, ConditionSetSet > ConflictSets`
- typedef `std::vector< std::pair< ConditionList, bool > > SubstitutionResult`
- typedef `std::vector< SubstitutionResult > SubstitutionResults`
- typedef `std::vector< std::pair< unsigned, unsigned > > SubResultCombination`
- typedef `smrat::vb::VariableBounds< const Condition * > VariableBoundsCond`

### Public Member Functions

- `State (carl::IDPool *_conditionIdAllocator, bool _withVariableBounds)`

*Constructs an empty state (no conditions yet) being the root (hence neither a substitution) of the state tree which is going to be formed when applying the satisfiability check based on virtual substitution.*
- `State (State *const _father, const Substitution &_substitution, carl::IDPool *_conditionIdAllocator, bool _← withVariableBounds)`

*Constructs a state being a child of the given state and containing the given substitution, which maps from the index variable of the given state.*
- `State (const State &)=delete`
- `~State ()`

*Destructor.*
- `bool isRoot () const`
- `bool cannotBeSolved (bool _lazy) const`
- `bool & rCannotBeSolved ()`
- `bool & rCannotBeSolvedLazy ()`
- `bool markedAsDeleted () const`
- `bool & rMarkedAsDeleted ()`
- `bool hasChildrenToInsert () const`
- `bool & rHasChildrenToInsert ()`
- `carl::Variable::Arg index () const`
- `size_t valuation () const`
- `size_t backendCallValuation () const`
- `size_t id () const`
- `size_t & rID ()`
- `std::list< State * > & rChildren ()`
- `const std::list< State * > & children () const`
- `State * pFather () const`
- `const State & father () const`
- `State & rFather ()`
- `ConflictSets & rConflictSets ()`
- `const ConflictSets & conflictSets () const`
- `bool & rHasRecentlyAddedConditions ()`
- `bool hasRecentlyAddedConditions () const`
- `bool & rInconsistent ()`

- bool `isInconsistent () const`
- `ConditionList & rConditions ()`
- const `ConditionList & conditions () const`
- `Substitution & rSubstitution ()`
- const `Substitution & substitution () const`
- `SubstitutionResults & rSubstitutionResults ()`
- const `SubstitutionResults & substitutionResults () const`
- `SubResultCombination & rSubResultCombination ()`
- const `SubResultCombination & subResultCombination () const`
- const `Substitution * pSubstitution () const`
- bool `conditionsSimplified () const`
- bool `subResultsSimplified () const`
- bool & `rSubResultsSimplified ()`
- bool `takeSubResultCombAgain () const`
- bool & `rTakeSubResultCombAgain ()`
- bool `tryToRefreshIndex () const`
- bool `hasSubResultsCombination () const`
- bool `hasSubstitutionResults () const`
- bool `unfinished () const`
- `Type type () const`
- `Type & rType ()`
- const `Condition * pOriginalCondition () const`
- const `Condition & originalCondition () const`
- const `carl::PointerSet< Condition > & tooHighDegreeConditions () const`
- `carl::PointerSet< Condition > & rTooHighDegreeConditions ()`
- const `VariableBoundsCond & variableBounds () const`
- `VariableBoundsCond & rVariableBounds ()`
- void `setOriginalCondition (const Condition *_pOCondition)`

*Sets the pointer of the original condition of this state to the given pointer to a condition.*

- size\_t `currentRangeSize () const`
- void `resetCurrentRangeSize ()`
- const `smrat::Rational & minIntTestCandidate () const`
- const `smrat::Rational & maxIntTestCandidate () const`
- bool `hasInfinityChild () const`
- void `resetInfinityChild (const State *_state)`
- void `setInfinityChild (State *_state)`
- void `resetCannotBeSolvedLazyFlags ()`
- unsigned `treeDepth () const`
- bool `substitutionApplicable () const`

*Checks if a substitution can be applied.*

- bool `substitutionApplicable (const smrat::ConstraintT &_constraint) const`

*Checks if the substitution of this state can be applied to the given constraint.*

- bool `hasNoninvolvedCondition () const`

*Checks whether a condition exists, which was not involved in an elimination step.*

- bool `allTestCandidatesInvalidated (const Condition *_condition) const`
- bool `hasChildWithID () const`

*Checks whether a child exists, which has no ID (!=0).*

- bool `containsState (const State *_state) const`
- bool `hasOnlyInconsistentChildren () const`

*Checks whether a child exists, which is not yet marked as inconsistent.*

- bool `occursInEquation (const carl::Variable &) const`

*Checks whether the given variable occurs in a equation.*

- bool `hasFurtherUncheckedTestCandidates () const`

- Checks whether there exist more than one test candidate, which has still not been checked.
- void `variables` (carl::Variables &\_variables) const
  - Finds the variables, which occur in this state.*
- unsigned `numberOfNodes` () const
  - Determines the number of nodes in the tree with this state as root.*
- `State & root` ()
  - `bool getNextIntTestCandidate (smrat::Rational &_nextIntTestCandidate, size_t _maxIntRange)`
  - `bool updateIntTestCandidates ()`
  - `bool unfinishedAncestor (State *&_unfinAnt)`
    - Determines (if it exists) a ancestor node, which is unfinished, that is it has still substitution results to consider.*
- `bool bestCondition (const Condition *&_bestCondition, size_t _numberOfAllVariables, bool _preferEquation)`
  - Determines the most adequate condition and in it the most adequate variable in the state to generate test candidates for.*
- ConditionList::iterator `constraintExists (const smrat::ConstraintT &_constraint)`
  - Checks if the given constraint already exists as condition in the state.*
- void `simplify (ValuationMap &_ranking)`
  - Cleans up all conditions in this state according to comparison between the corresponding constraints.*
- `bool simplify (ConditionList &_conditionVectorToSimplify, ConditionSetSet &_conflictSet, ValuationMap &_ranking, bool _stateConditions=false)`
  - Simplifies the given conditions according to comparison between the corresponding constraints.*
- void `setIndex (const carl::Variable &_index)`
  - Sets the index of this state.*
- void `addConflictSet (const Substitution * _substitution, ConditionSetSet &&_condSetSet)`
  - Adds a conflict set to the map of substitutions to conflict sets.*
- void `addConflicts (const Substitution * _substitution, ConditionSetSet &&_condSetSet)`
  - Adds all conflicts to all sets of the conflict set of the given substitution.*
- void `resetConflictSets ()`
  - Clears the conflict sets.*
- `bool updateOCondsOfSubstitutions (const Substitution &_substitution, std::vector< State * > &_reactivatedStates)`
  - Updates the original conditions of substitutions having the same test candidate as the given.*
- void `addSubstitutionResults (std::vector< DisjunctionOfConditionConjunctions > &&_disjunctionsOfCond Conj)`
  - Adds the given substitution results to this state.*
- bool `extendSubResultCombination ()`
  - Extends the currently considered combination of conjunctions in the substitution results.*
- bool `nextSubResultCombination ()`
  - If the state contains a substitution result, which is a conjunction of disjunctions of conjunctions of conditions, this method sets the current combination to the disjunctions to the next combination.*
- size\_t `getNumberOfCurrentSubresultCombination () const`
- ConditionList `getCurrentSubresultCombination () const`
  - Gets the current substitution result combination as condition vector.*
- bool `refreshConditions (ValuationMap &_ranking)`
  - Determines the condition vector corresponding to the current combination of the conjunctions of conditions.*
- void `initConditionFlags ()`
  - Sets all flags of the conditions to true, if it contains the variable given by the states index.*
- bool `initIndex (const carl::Variables &_allVariables, const smrat::VariableValuationStrategy &_vvstrat, bool _preferEquation, bool _tryDifferentVarOrder=false, bool _useFixedVariableOrder=false)`
  - Sets, if it has not already happened, the index of the state to the name of the most adequate variable.*
- void `bestConstraintValuation (const std::vector< std::pair< carl::Variable, std::multiset< double >>> &_varVals)`

- void `averageConstraintValuation` (const std::vector< std::pair< carl::Variable, std::multiset< double >>> &\_varVals)
- void `worstConstraintValuation` (const std::vector< std::pair< carl::Variable, std::multiset< double >>> &\_varVals)
- void `addCondition` (const smrat::ConstraintT &\_constraint, const carl::PointerSet< Condition > &\_originalConditions, size\_t \_valuation, bool \_recentlyAdded, ValuationMap &\_ranking)  
*Adds a constraint to the conditions of this state.*
- bool `checkSubresultCombinations` () const  
*This is just for debug purpose.*
- int `deleteOrigins` (carl::PointerSet< Condition > &\_originsToDelete, ValuationMap &\_ranking)  
*Removes everything in this state originated by the given vector of conditions.*
- void `deleteOriginsFromChildren` (carl::PointerSet< Condition > &\_originsToDelete, ValuationMap &\_ranking)  
*Deletes everything originated by the given conditions in the children of this state.*
- void `deleteOriginsFromConflictSets` (carl::PointerSet< Condition > &\_originsToDelete, bool \_originsAreCurrentConditions)  
*Deletes everything originated by the given conditions in the conflict sets of this state.*
- void `deleteOriginsFromSubstitutionResults` (carl::PointerSet< Condition > &\_originsToDelete)  
*Deletes everything originated by the given conditions in the substitution results of this state.*
- void `deleteConditions` (carl::PointerSet< Condition > &\_conditionsToDelete, ValuationMap &\_ranking)  
*Delete everything originated by the given conditions from the entire subtree with this state as root.*
- std::vector< State \* > `addChild` (const Substitution &\_substitution)  
*Adds a state as child to this state with the given substitution.*
- void `updateValuation` (bool \_lazy)  
*Updates the valuation of this state.*
- void `updateBackendCallValuation` ()  
*Valuates the state's currently considered conditions according to a backend call.*
- void `passConflictToFather` (bool \_checkConflictForSideCondition, bool \_useBackjumping=true, bool \_includeInconsistentTestCandidates=false)  
*Passes the original conditions of the covering set of the conflicts of this state to its father.*
- bool `hasLocalConflict` ()  
*Checks whether the currently considered conditions, which have been considered for test candidate construction, form already a conflict.*
- bool `checkTestCandidatesForBounds` ()  
*Checks whether the test candidate of this state is valid against the variable intervals in the father of this state.*
- std::vector< smrat::DoubleInterval > `solutionSpace` (carl::PointerSet< Condition > &\_conflictReason) const  
*Determines the solution space of the test candidate of this state regarding to the variable bounds of the father.*
- bool `hasRootsInVariableBounds` (const Condition \* \_condition, bool \_useSturmSequence)  
*Checks whether there are no zeros for the left-hand side of the constraint of the given condition.*
- void `print` (const std::string \_initiation="\*\*\*", std::ostream &\_out=std::cout) const  
*Prints the conditions, the substitution and the substitution results of this state and all its children.*
- void `printAlone` (const std::string \_initiation="\*\*\*", std::ostream &\_out=std::cout) const  
*Prints the conditions, the substitution and the substitution results of this state.*
- void `printConditions` (const std::string \_initiation="\*\*\*", std::ostream &\_out=std::cout, bool=false) const  
*Prints the conditions of this state.*
- void `printSubstitutionResults` (const std::string \_initiation="\*\*\*", std::ostream &\_out=std::cout) const  
*Prints the substitution results of this state.*
- void `printSubstitutionResultCombination` (const std::string \_initiation="\*\*\*", std::ostream &\_out=std::cout) const  
*Prints the combination of substitution results used in this state.*
- void `printSubstitutionResultCombinationAsNumbers` (const std::string \_initiation="\*\*\*", std::ostream &\_out=std::cout) const  
*Prints the combination of substitution results, expressed in numbers, used in this state.*
- void `printConflictSets` (const std::string \_initiation="\*\*\*", std::ostream &\_out=std::cout) const  
*Prints the conflict sets of this state.*

## Static Public Member Functions

- static void `removeStatesFromRanking` (const `State` &`toRemove`, `ValuationMap` &`_ranking`)
- static size\_t `coveringSet` (const `ConditionSetSet` &`_conflictSets`, carl::PointerSet< `Condition` > &`_minCovSet`, unsigned `_currentTreeDepth`)

*Finds a covering set of a vector of sets of sets due to some heuristics.*

### 0.15.520.1 Member Typedef Documentation

#### 0.15.520.1.1 `ConflictSets` `typedef carl::FastPointerMapB<Substitution, ConditionSetSet>` `smtrat::vs::State::Co`

#### 0.15.520.1.2 `SubResultCombination` `typedef std::vector<std::pair< unsigned, unsigned> >` `smtrat::vs::State::Su`

#### 0.15.520.1.3 `SubstitutionResult` `typedef std::vector<std::pair< ConditionList, bool> >` `smtrat::vs::State::Subst`

#### 0.15.520.1.4 `SubstitutionResults` `typedef std::vector<SubstitutionResult>` `smtrat::vs::State::SubstitutionResults`

#### 0.15.520.1.5 `VariableBoundsCond` `typedef smtrat::vb::VariableBounds<const Condition*>` `smtrat::vs::State::Vari`

### 0.15.520.2 Member Enumeration Documentation

#### 0.15.520.2.1 `Type` `enum smtrat::vs::State::Type`

Enumerator

|                            |  |
|----------------------------|--|
| TEST_CANDIDATE_TO_GENERATE |  |
| SUBSTITUTION_TO_APPLY      |  |
| COMBINE_SUBRESULTS         |  |

### 0.15.520.3 Constructor & Destructor Documentation

#### 0.15.520.3.1 `State()` [1/3] `smtrat::vs::State::State (` `carl::IDPool * _conditionIdAllocator,` `bool _withVariableBounds )`

Constructs an empty state (no conditions yet) being the root (hence neither a substitution) of the state tree which is going to be formed when applying the satisfiability check based on virtual substitution.

Parameters

|                                  |                                                                              |
|----------------------------------|------------------------------------------------------------------------------|
| <code>_withVariableBounds</code> | A flag that indicates whether to use optimizations based on variable bounds. |
|----------------------------------|------------------------------------------------------------------------------|

#### 0.15.520.3.2 `State()` [2/3] `smtrat::vs::State::State (` `State *const _father,`

```
const Substitution & _substitution,
carl::IDPool * _conditionIdAllocator,
bool _withVariableBounds)
```

Constructs a state being a child of the given state and containing the given substitution, which maps from the index variable of the given state.

#### Parameters

|                                  |                                                                              |
|----------------------------------|------------------------------------------------------------------------------|
| <code>_father</code>             | The father of the state to be constructed.                                   |
| <code>_substitution</code>       | The substitution of the state to be constructed.                             |
| <code>_withVariableBounds</code> | A flag that indicates whether to use optimizations based on variable bounds. |

**0.15.520.3.3 State()** [3/3] smtrat::vs::State::State (

```
const State &) [delete]
```

**0.15.520.3.4 ~State()** smtrat::vs::State::~State ( )  
Destructor.

### 0.15.520.4 Member Function Documentation

**0.15.520.4.1 addChild()** std::vector< `State` \* > smtrat::vs::State::addChild (

```
const Substitution & _substitution)
```

Adds a state as child to this state with the given substitution.

#### Parameters

|                            |                                             |
|----------------------------|---------------------------------------------|
| <code>_substitution</code> | The substitution to generate the state for. |
|----------------------------|---------------------------------------------|

#### Returns

true, if a state has been added as child to this state; false, otherwise.

**0.15.520.4.2 addCondition()** void smtrat::vs::State::addCondition (

```
const smtrat::ConstraintT & _constraint,
const carl::PointerSet< Condition > & _originalConditions,
size_t _valuation,
bool _recentlyAdded,
ValuationMap & _ranking)
```

Adds a constraint to the conditions of this state.

#### Parameters

|                                  |                                                                                          |
|----------------------------------|------------------------------------------------------------------------------------------|
| <code>_constraint</code>         | The constraint of the condition to add.                                                  |
| <code>_originalConditions</code> | The original conditions of the condition to add.                                         |
| <code>_valuation</code>          | The valuation of the condition to add.                                                   |
| <code>_recentlyAdded</code>      | Is the condition a recently added one. @sideeffect The state can obtain a new condition. |

```
0.15.520.4.3 addConflicts() void smtrat::vs::State::addConflicts (
 const Substitution * _substitution,
 ConditionSetSet && _condSetSet)
```

Adds all conflicts to all sets of the conflict set of the given substitution.

#### Parameters

|                            |                                                                                                                    |
|----------------------------|--------------------------------------------------------------------------------------------------------------------|
| <code>_substitution</code> | The corresponding substitution generated the conflict. (NULL in the case a detected conflict without substitution) |
| <code>_condSetSet</code>   | The conflicts to add.                                                                                              |

```
0.15.520.4.4 addConflictSet() void smtrat::vs::State::addConflictSet (
 const Substitution * _substitution,
 ConditionSetSet && _condSetSet)
```

Adds a conflict set to the map of substitutions to conflict sets.

#### Parameters

|                            |                                                                                                                    |
|----------------------------|--------------------------------------------------------------------------------------------------------------------|
| <code>_substitution</code> | The corresponding substitution generated the conflict. (NULL in the case a detected conflict without substitution) |
| <code>_condSetSet</code>   | The conflicts to add.                                                                                              |

```
0.15.520.4.5 addSubstitutionResults() void smtrat::vs::State::addSubstitutionResults (
 std::vector< DisjunctionOfConditionConjunctions > && _disjunctionsOfCondConj)
```

Adds the given substitution results to this state.

#### Parameters

|                                      |                                                                                           |
|--------------------------------------|-------------------------------------------------------------------------------------------|
| <code>_disjunctionsOfCondConj</code> | The substitution results given by a vector of disjunctions of conjunctions of conditions. |
|--------------------------------------|-------------------------------------------------------------------------------------------|

```
0.15.520.4.6 allTestCandidatesInvalidated() bool smtrat::vs::State::allTestCandidatesInvalidated (
 const Condition * _condition) const
```

#### Parameters

|                |                                                                                              |
|----------------|----------------------------------------------------------------------------------------------|
| <code>A</code> | condition, which is one of the currently considered constraints to check for satisfiability. |
|----------------|----------------------------------------------------------------------------------------------|

#### Returns

true, if all test candidates provided by the given condition are no solution of the currently considered conjunction of constraints to check for satisfiability.

```
0.15.520.4.7 averageConstraintValuation() void smtrat::vs::State::averageConstraintValuation (
 const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _←
 varVals)
```

```
0.15.520.4.8 backendCallValuation() size_t smtrat::vs::State::backendCallValuation () const
[inline]
```

**Returns**

The valuation of the state's considered conditions for a possible backend call.

```
0.15.520.4.9 bestCondition() bool smtrat::vs::State::bestCondition (
 const Condition *& _bestCondition,
 size_t _numberOfAllVariables,
 bool _preferEquation)
```

Determines the most adequate condition and in it the most adequate variable in the state to generate test candidates for.

**Parameters**

|                                    |                                                                             |
|------------------------------------|-----------------------------------------------------------------------------|
| <code>_bestCondition</code>        | The most adequate condition to be the next test candidate provider.         |
| <code>_numberOfAllVariables</code> | The number of all globally known variables.                                 |
| <code>_preferEquation</code>       | A flag that indicates to prefer equations in the heuristics of this method. |

**Returns**

true, if it has a condition and a variable in it to generate test candidates for; false, otherwise.

```
0.15.520.4.10 bestConstraintValuation() void smtrat::vs::State::bestConstraintValuation (
 const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _←
 varVals)
```

```
0.15.520.4.11 cannotBeSolved() bool smtrat::vs::State::cannotBeSolved (
 bool _lazy) const [inline]
```

**Returns**

A constant reference to the flag indicating whether a condition with too high degree for the virtual substitution method must be considered.

```
0.15.520.4.12 checkSubresultCombinations() bool smtrat::vs::State::checkSubresultCombinations (
) const
```

This is just for debug purpose.

**Returns**

true, if no subresult combination is illegal.

```
0.15.520.4.13 checkTestCandidatesForBounds() bool smtrat::vs::State::checkTestCandidatesFor←
Bounds ()
```

Checks whether the test candidate of this state is valid against the variable intervals in the father of this state.

**Returns**

true, if the test candidate of this state is valid against the variable intervals; false, otherwise.

**0.15.520.4.14 children()** const std::list<[State](#)\*>& smtrat::vs::State::children () const [inline]**Returns**

A constant reference to the vector of the children of this state.

**0.15.520.4.15 conditions()** const [ConditionList](#)& smtrat::vs::State::conditions () const [inline]**Returns**

A constant reference to the currently considered conditions of this state.

**0.15.520.4.16 conditionsSimplified()** bool smtrat::vs::State::conditionsSimplified () const [inline]**Returns**

true, if the considered conditions of this state are already simplified.

**0.15.520.4.17 conflictSets()** const [ConflictSets](#)& smtrat::vs::State::conflictSets () const [inline]**Returns**

A constant reference to the conflict sets of this state.

**0.15.520.4.18 constraintExists()** ConditionList::iterator smtrat::vs::State::constraintExists (const [smtrat::ConstraintT](#) & *\_constraint*)

Checks if the given constraint already exists as condition in the state.

**Parameters**

|                    |                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------|
| <i>_constraint</i> | The constraint, for which we want to know, if it already exists as condition in the state. |
|--------------------|--------------------------------------------------------------------------------------------|

**Returns**

An iterator to the condition, which involves the constraint or an iterator to the end of the vector of conditions of this state.

**0.15.520.4.19 containsState()** bool smtrat::vs::State::containsState (const [State](#) \* *\_state*) const**Returns**

true, if the given state is a node in the state tree starting at this node; false, otherwise.

**0.15.520.4.20 coveringSet()** size\_t smtrat::vs::State::coveringSet (const [ConditionSetSet](#) & *\_conflictSets*, carl::PointerSet< [Condition](#) > & *\_minCovSet*, unsigned *\_currentTreeDepth*) [static]

Finds a covering set of a vector of sets of sets due to some heuristics.

**Parameters**

|                                |                                                                                   |
|--------------------------------|-----------------------------------------------------------------------------------|
| <code>_conflictSets</code>     | The vector of sets of sets, for which the method finds all minimum covering sets. |
| <code>_minCovSet</code>        | The found minimum covering set.                                                   |
| <code>_currentTreeDepth</code> | The tree depth of the state from which this method is invoked.                    |

**Returns**

The greatest level, where a condition of the covering set has been created.

**0.15.520.4.21 `currentRangeSize()`** `size_t smtrat::vs::State::currentRangeSize ( ) const [inline]`

```
0.15.520.4.22 deleteConditions() void smtrat::vs::State::deleteConditions (
 carl::PointerSet< Condition > & _conditionsToDelete,
 ValuationMap & _ranking)
```

Delete everything originated by the given conditions from the entire subtree with this state as root.

**Parameters**

|                                  |                           |
|----------------------------------|---------------------------|
| <code>_conditionsToDelete</code> | The conditions to delete. |
|----------------------------------|---------------------------|

```
0.15.520.4.23 deleteOrigins() int smtrat::vs::State::deleteOrigins (
 carl::PointerSet< Condition > & _originsToDelete,
 ValuationMap & _ranking)
```

Removes everything in this state originated by the given vector of conditions.

**Parameters**

|                               |                                                                                                      |
|-------------------------------|------------------------------------------------------------------------------------------------------|
| <code>_originsToDelete</code> | The conditions for which everything in this state which has been originated by them must be removed. |
|-------------------------------|------------------------------------------------------------------------------------------------------|

**Returns**

0, if this state got invalid and must be deleted afterwards; -1, if this state got invalid and must be deleted afterwards and made other states unnecessary to consider; 1, otherwise.

```
0.15.520.4.24 deleteOriginsFromChildren() void smtrat::vs::State::deleteOriginsFromChildren (
 carl::PointerSet< Condition > & _originsToDelete,
 ValuationMap & _ranking)
```

Deletes everything originated by the given conditions in the children of this state.

**Parameters**

|                               |                                                                  |
|-------------------------------|------------------------------------------------------------------|
| <code>_originsToDelete</code> | The condition for which to delete everything originated by them. |
|-------------------------------|------------------------------------------------------------------|

```
0.15.520.4.25 deleteOriginsFromConflictSets() void smtrat::vs::State::deleteOriginsFromConflictSets (
```

```
carl::PointerSet< Condition > & _originsToDelete,
bool _originsAreCurrentConditions)
```

Deletes everything originated by the given conditions in the conflict sets of this state.

#### Parameters

|                                           |                                                                                                                                                                                                          |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_originsToDelete</code>             | The condition for which to delete everything originated by them.                                                                                                                                         |
| <code>_originsAreCurrentConditions</code> | A flag indicating whether the origins to remove stem from conditions of the currently considered condition vector of the father of this state and not, e.g., from somewhere in its substitution results. |

**0.15.520.4.26 `deleteOriginsFromSubstitutionResults()`** void smtrat::vs::State::deleteOriginsFromSubstitutionResults (

```
carl::PointerSet< Condition > & _originsToDelete)
```

Deletes everything originated by the given conditions in the substitution results of this state.

#### Parameters

|                               |                                                                   |
|-------------------------------|-------------------------------------------------------------------|
| <code>_originsToDelete</code> | The conditions for which to delete everything originated by them. |
|-------------------------------|-------------------------------------------------------------------|

**0.15.520.4.27 `extendSubResultCombination()`** bool smtrat::vs::State::extendSubResultCombination ( )

Extends the currently considered combination of conjunctions in the substitution results.

**0.15.520.4.28 `father()`** const State& smtrat::vs::State::father ( ) const [inline]

#### Returns

A constant reference to the father of this state. This method fails if this state is the root.

**0.15.520.4.29 `getCurrentSubresultCombination()`** ConditionList smtrat::vs::State::getCurrentSubresultCombination ( ) const

Gets the current substitution result combination as condition vector.

#### Returns

The current substitution result combination as condition vector.

**0.15.520.4.30 `getNextIntTestCandidate()`** bool smtrat::vs::State::getNextIntTestCandidate (

```
smtrat::Rational & _nextIntTestCandidate,
```

```
size_t _maxIntRange)
```

**0.15.520.4.31 `getNumberOfCurrentSubresultCombination()`** size\_t smtrat::vs::State::getNumberOfCurrentSubresultCombination ( ) const

#### Returns

The number of the current substitution result combination.

**0.15.520.4.32 hasChildrenToInsert()** `bool smtrat::vs::State::hasChildrenToInsert () const [inline]`

Returns

A constant reference to the flag indicating whether there are states that are children of this state and must be still considered in the further progress of finding a solution.

**0.15.520.4.33 hasChildWithID()** `bool smtrat::vs::State::hasChildWithID () const`

Checks whether a child exists, which has no ID ( $\neq 0$ ).

Returns

true, if there exists a child with ID ( $\neq 0$ ); false, otherwise.

**0.15.520.4.34 hasFurtherUncheckedTestCandidates()** `bool smtrat::vs::State::hasFurtherUncheckedTestCandidates () const`

Checks whether there exist more than one test candidate, which has still not been checked.

Returns

true, if there exist more than one test candidate, which has still not been checked; false, otherwise.

**0.15.520.4.35 hasInfinityChild()** `bool smtrat::vs::State::hasInfinityChild () const [inline]`**0.15.520.4.36 hasLocalConflict()** `bool smtrat::vs::State::hasLocalConflict ()`

Checks whether the currently considered conditions, which have been considered for test candidate construction, form already a conflict.

Returns

true, if they form a conflict; false, otherwise.

**0.15.520.4.37 hasNoninvolvedCondition()** `bool smtrat::vs::State::hasNoninvolvedCondition () const`

Checks whether a condition exists, which was not involved in an elimination step.

Returns

true, if there exists a condition in the state, which was not already involved in an elimination step; false, otherwise.

**0.15.520.4.38 hasOnlyInconsistentChildren()** `bool smtrat::vs::State::hasOnlyInconsistentChildren () const`

Checks whether a child exists, which is not yet marked as inconsistent.

Returns

true, if there exists such a child; false, otherwise.

**0.15.520.4.39 hasRecentlyAddedConditions()** `bool smtrat::vs::State::hasRecentlyAddedConditions( ) const [inline]`

**Returns**

true, if this state has a recently added condition in its considered conditions.

**0.15.520.4.40 hasRootsInVariableBounds()** `bool smtrat::vs::State::hasRootsInVariableBounds( const Condition * _condition, bool _useSturmSequence )`

Checks whether there are no zeros for the left-hand side of the constraint of the given condition.

**Parameters**

|                         |                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------|
| <code>_condition</code> | The condition to check.                                                                    |
| <code>A</code>          | flag that indicates whether to include Sturm's sequences and root counting in this method. |

**Returns**

true, if the constraint of the left-hand side of the given condition has no roots in the variable bounds of this state; false, otherwise.

**0.15.520.4.41 hasSubResultsCombination()** `bool smtrat::vs::State::hasSubResultsCombination( ) const [inline]`

**Returns**

true, if this state has a substitution result combination.

**0.15.520.4.42 hasSubstitutionResults()** `bool smtrat::vs::State::hasSubstitutionResults( ) const [inline]`

**Returns**

true, if this state has substitution results.

**0.15.520.4.43 id()** `size_t smtrat::vs::State::id( ) const [inline]`

**Returns**

The id of this state.

**0.15.520.4.44 index()** `carl::Variable::Arg smtrat::vs::State::index( ) const [inline]`

**Returns**

A constant reference to the variable which is going to be eliminated from this state's considered conditions.  
Note, if there is no variable decided to be eliminated, this method will fail.

**0.15.520.4.45 initConditionFlags()** `void smtrat::vs::State::initConditionFlags( )`

Sets all flags of the conditions to true, if it contains the variable given by the states index.

```
0.15.520.4.46 initIndex() bool smtrat::vs::State::initIndex (
 const carl::Variables & _allVariables,
 const smtrat::VariableValuationStrategy & _vvstrat,
 bool _preferEquation,
 bool _tryDifferentVarOrder = false,
 bool _useFixedVariableOrder = false)
```

Sets, if it has not already happened, the index of the state to the name of the most adequate variable.

Which variable is taken depends on heuristics.

#### Parameters

|                                    |                                                                             |
|------------------------------------|-----------------------------------------------------------------------------|
| <code>_allVariables</code>         | All globally known variables.                                               |
| <code>_vvstrat</code>              | The strategy according which we choose the variable.                        |
| <code>_preferEquation</code>       | A flag that indicates to prefer equations in the heuristics of this method. |
| <code>_tryDifferentVarOrder</code> |                                                                             |

```
0.15.520.4.47 isInconsistent() bool smtrat::vs::State::isInconsistent () const [inline]
```

#### Returns

true if this state's considered conditions are inconsistent; false, otherwise, which does not necessarily mean that this state is consistent.

```
0.15.520.4.48 isRoot() bool smtrat::vs::State::isRoot () const [inline]
```

#### Returns

The root of the state tree where this state is part from.

```
0.15.520.4.49 markedAsDeleted() bool smtrat::vs::State::markedAsDeleted () const [inline]
```

#### Returns

A constant reference to the flag indicating whether this state is marked as deleted, which means that there is no need to consider it in the further progress of finding a solution.

```
0.15.520.4.50 maxIntTestCandidate() const smtrat::Rational& smtrat::vs::State::maxIntTest<->
Candidate () const [inline]
```

```
0.15.520.4.51 minIntTestCandidate() const smtrat::Rational& smtrat::vs::State::minIntTest<->
Candidate () const [inline]
```

```
0.15.520.4.52 nextSubResultCombination() bool smtrat::vs::State::nextSubResultCombination ()
```

If the state contains a substitution result, which is a conjunction of disjunctions of conjunctions of conditions, this method sets the current combination to the disjunctions to the next combination.

#### Returns

true, if there is a next combination; false, otherwise.

**0.15.520.4.53 `numberOfNodes()`** `unsigned smtrat::vs::State::numberOfNodes ( ) const`  
Determines the number of nodes in the tree with this state as root.

**Returns**

The number of nodes in the tree with this state as root.

**0.15.520.4.54 `occursInEquation()`** `bool smtrat::vs::State::occursInEquation (`  
`const carl::Variable & _variable ) const`  
Checks whether the given variable occurs in a equation.

**Returns**

true, if the given variable occurs in a equation; false, otherwise.

**0.15.520.4.55 `originalCondition()`** `const Condition& smtrat::vs::State::originalCondition ( ) const [inline]`

**Returns**

A reference to the original conditions of this state. Note that if there is no such condition, this method fails. For further information see the description of the corresponding member.

**0.15.520.4.56 `passConflictToFather()`** `void smtrat::vs::State::passConflictToFather (`  
`bool _checkConflictForSideCondition,`  
`bool _useBackjumping = true,`  
`bool _includeInconsistentTestCandidates = false )`

Passes the original conditions of the covering set of the conflicts of this state to its father.

**Parameters**

|                                                 |                                                                                                                                                        |
|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_checkConflictForSideCondition</code>     | If this flag is set to false it might save some computations but leads to an over-approximative behavior. Recommendation: set it to true.              |
| <code>_includeInconsistentTestCandidates</code> | If this flag is set to true the conflict analysis takes also the invalid test candidates' conflict sets into account. Recommendation: set it to false. |
| <code>_useBackjumping</code>                    | If true, we use the backjumping technique.                                                                                                             |

**0.15.520.4.57 `pFather()`** `State* smtrat::vs::State::pFather ( ) const [inline]`

**Returns**

A pointer to the father of this state. It is NULL if this state is the root.

**0.15.520.4.58 `pOriginalCondition()`** `const Condition* smtrat::vs::State::pOriginalCondition ( ) const [inline]`

**Returns**

A pointer to the original condition of this state or NULL, if there is no such condition. For further information see the description of the corresponding member.

```
0.15.520.4.59 print() void smtrat::vs::State::print (
 const std::string _initiation = "***",
 std::ostream & _out = std::cout) const
```

Prints the conditions, the substitution and the substitution results of this state and all its children.

#### Parameters

|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <code>_initiation</code> | A string which is printed in the beginning of every row. |
| <code>_out</code>        | The stream to print on.                                  |

```
0.15.520.4.60 printAlone() void smtrat::vs::State::printAlone (
 const std::string _initiation = "***",
 std::ostream & _out = std::cout) const
```

Prints the conditions, the substitution and the substitution results of this state.

#### Parameters

|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <code>_initiation</code> | A string which is printed in the beginning of every row. |
| <code>_out</code>        | The stream to print on.                                  |

```
0.15.520.4.61 printConditions() void smtrat::vs::State::printConditions (
 const std::string _initiation = "***",
 std::ostream & _out = std::cout,
 bool _onlyAsConstraints = false) const
```

Prints the conditions of this state.

#### Parameters

|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <code>_initiation</code> | A string which is printed in the beginning of every row. |
| <code>_out</code>        | The stream to print on.                                  |

```
0.15.520.4.62 printConflictSets() void smtrat::vs::State::printConflictSets (
 const std::string _initiation = "***",
 std::ostream & _out = std::cout) const
```

Prints the conflict sets of this state.

#### Parameters

|                          |                                                          |
|--------------------------|----------------------------------------------------------|
| <code>_initiation</code> | A string which is printed in the beginning of every row. |
| <code>_out</code>        | The stream to print on.                                  |

```
0.15.520.4.63 printSubstitutionResultCombination() void smtrat::vs::State::printSubstitution<-
ResultCombination (
 const std::string _initiation = "***",
 std::ostream & _out = std::cout) const
```

Prints the combination of substitution results used in this state.

## Parameters

|                    |                                                          |
|--------------------|----------------------------------------------------------|
| <i>_initiation</i> | A string which is printed in the beginning of every row. |
| <i>_out</i>        | The stream to print on.                                  |

**0.15.520.4.64 printSubstitutionResultCombinationAsNumbers()** void smtrat::vs::State::printSubstitutionResultCombinationAsNumbers (

```
 const std::string _initiation = "***",
 std::ostream & _out = std::cout) const
```

Prints the combination of substitution results, expressed in numbers, used in this state.

## Parameters

|                    |                                                          |
|--------------------|----------------------------------------------------------|
| <i>_initiation</i> | A string which is printed in the beginning of every row. |
| <i>_out</i>        | The stream to print on.                                  |

**0.15.520.4.65 printSubstitutionResults()** void smtrat::vs::State::printSubstitutionResults (

```
 const std::string _initiation = "***",
 std::ostream & _out = std::cout) const
```

Prints the substitution results of this state.

## Parameters

|                    |                                                          |
|--------------------|----------------------------------------------------------|
| <i>_initiation</i> | A string which is printed in the beginning of every row. |
| <i>_out</i>        | The stream to print on.                                  |

**0.15.520.4.66 pSubstitution()** const Substitution\* smtrat::vs::State::pSubstitution ( ) const

[inline]

## Returns

A pointer to the substitution of this state, which is NULL, if this state is the root.

**0.15.520.4.67 rCannotBeSolved()** bool& smtrat::vs::State::rCannotBeSolved ( ) [inline]

## Returns

A reference to the flag indicating whether a condition with too high degree for the virtual substitution method must be considered.

**0.15.520.4.68 rCannotBeSolvedLazy()** bool& smtrat::vs::State::rCannotBeSolvedLazy ( ) [inline]

## Returns

A reference to the flag indicating whether a condition with too high degree for the virtual substitution method must be considered.

**0.15.520.4.69 rChildren()** `std::list<State*>& smtrat::vs::State::rChildren () [inline]`

Returns

A reference to the vector of the children of this state.

**0.15.520.4.70 rConditions()** `ConditionList& smtrat::vs::State::rConditions () [inline]`

Returns

A reference to the currently considered conditions of this state.

**0.15.520.4.71 rConflictSets()** `ConflictSets& smtrat::vs::State::rConflictSets () [inline]`

Returns

A reference to the conflict sets of this state.

**0.15.520.4.72 refreshConditions()** `bool smtrat::vs::State::refreshConditions ( ValuationMap & _ranking )`

Determines the condition vector corresponding to the current combination of the conjunctions of conditions.

Returns

true, if there has been a change in the currently considered condition vector. false, otherwise.

**0.15.520.4.73 removeStatesFromRanking()** `void smtrat::vs::State::removeStatesFromRanking ( const State & toRemove, ValuationMap & _ranking ) [static]`

**0.15.520.4.74 resetCannotBeSolvedLazyFlags()** `void smtrat::vs::State::resetCannotBeSolvedLazyFlags ()`

**0.15.520.4.75 resetConflictSets()** `void smtrat::vs::State::resetConflictSets ()`  
Clears the conflict sets.

**0.15.520.4.76 resetCurrentRangeSize()** `void smtrat::vs::State::resetCurrentRangeSize () [inline]`

**0.15.520.4.77 resetInfinityChild()** `void smtrat::vs::State::resetInfinityChild ( const State * _state ) [inline]`

**0.15.520.4.78 rFather()** `State& smtrat::vs::State::rFather () [inline]`

Returns

A reference to the father of this state. This method fails if this state is the root.

**0.15.520.4.79 rHasChildrenToInsert()** `bool& smtrat::vs::State::rHasChildrenToInsert () [inline]`

Returns

A reference to the flag indicating whether there are states that are children of this state and must be still considered in the further progress of finding a solution.

**0.15.520.4.80 rHasRecentlyAddedConditions()** `bool& smtrat::vs::State::rHasRecentlyAddedConditions () [inline]`

Returns

A reference to the flag indicating that this state has a recently added condition in its considered conditions.

**0.15.520.4.81 rID()** `size_t& smtrat::vs::State::rID () [inline]`

Returns

A reference to the id of this state.

**0.15.520.4.82 rInconsistent()** `bool& smtrat::vs::State::rInconsistent () [inline]`

Returns

A reference to the flag indicating whether this state's considered conditions are inconsistent. Note that if it is false, it does not necessarily mean that this state is consistent.

**0.15.520.4.83 rMarkedAsDeleted()** `bool& smtrat::vs::State::rMarkedAsDeleted () [inline]`

Returns

A reference to the flag indicating whether this state is marked as deleted, which means that there is no need to consider it in the further progress of finding a solution.

**0.15.520.4.84 root()** `State & smtrat::vs::State::root ()`

Returns

The root of the tree, in which this state is located.

**0.15.520.4.85 rSubResultCombination()** `SubResultCombination& smtrat::vs::State::rSubResultCombination () [inline]`

Returns

A reference to the current combination of conjunction of constraints within the substitution results of this state. Make sure that a substitution result combination exists when calling this method. (using, e.g., [hasSubResultsCombination\(\)](#))

**0.15.520.4.86 rSubResultsSimplified()** `bool& smtrat::vs::State::rSubResultsSimplified ()` [inline]

Returns

A reference to the flag indicating whether the substitution results of this state are already simplified.

**0.15.520.4.87 rSubstitution()** `Substitution& smtrat::vs::State::rSubstitution ()` [inline]

Returns

A reference to the substitution this state considers. Note that this method fails, if this state is the root.

**0.15.520.4.88 rSubstitutionResults()** `SubstitutionResults& smtrat::vs::State::rSubstitutionResults ()` [inline]

Returns

A reference to the substitution results of this state. Make sure that the substitution results exist when calling this method. (using, e.g., `hasSubstitutionResults()`)

**0.15.520.4.89 rTakeSubResultCombAgain()** `bool& smtrat::vs::State::rTakeSubResultCombAgain ()` [inline]

Returns

A reference to the flag indicating whether the current substitution result combination has to be consider once again before considering the next one.

**0.15.520.4.90 rTooHighDegreeConditions()** `carl::PointerSet<Condition>& smtrat::vs::State::rTooHighDegreeConditions ()` [inline]

Returns

A reference to the conditions of those this state currently considers, which have a too high degree to be tackled of the virtual substitution method.

**0.15.520.4.91 rType()** `Type& smtrat::vs::State::rType ()` [inline]

Returns

A constant reference to the type of this state: This state has still a substitution to apply or either test candidates shall be generated or a new combination of the substitution results must be found in order to consider it next.

**0.15.520.4.92 rVariableBounds()** `VariableBoundsCond& smtrat::vs::State::rVariableBounds ()` [inline]

Returns

A reference to the object managing and storing the variable's bounds which were extracted from the currently considered conditions of this state.

**0.15.520.4.93 setIndex()** `void smtrat::vs::State::setIndex (const carl::Variable & _index )`

Sets the index of this state.

## Parameters

|                     |                                              |
|---------------------|----------------------------------------------|
| <code>_index</code> | The string to which the index should be set. |
|---------------------|----------------------------------------------|

**0.15.520.4.94 `setInfinityChild()`** `void smtrat::vs::State::setInfinityChild ( State * _state ) [inline]`

**0.15.520.4.95 `setOriginalCondition()`** `void smtrat::vs::State::setOriginalCondition ( const Condition * _pOCondition ) [inline]`

Sets the pointer of the original condition of this state to the given pointer to a condition.

## Parameters

|                           |                                               |
|---------------------------|-----------------------------------------------|
| <code>_pOCondition</code> | The pointer to set the original condition to. |
|---------------------------|-----------------------------------------------|

**0.15.520.4.96 `simplify()` [1/2]** `bool smtrat::vs::State::simplify ( ConditionList & _conditionVectorToSimplify, ConditionSetSet & _conflictSet, ValuationMap & _ranking, bool _stateConditions = false )`

Simplifies the given conditions according to comparison between the corresponding constraints.

## Parameters

|                                         |                                                                                                                                                                        |
|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_conditionVectorToSimplify</code> | The conditions to simplify. Note, that this method can change these conditions.                                                                                        |
| <code>_conflictSet</code>               | All conflicts this method detects in the given conditions are stored in here.                                                                                          |
| <code>_stateConditions</code>           | A flag indicating whether the conditions to simplify are from the considered condition vector of this state and not, e.g., from somewhere in its substitution results. |

## Returns

true, if the conditions are not obviously conflicting; false, otherwise.

**0.15.520.4.97 `simplify()` [2/2]** `void smtrat::vs::State::simplify ( ValuationMap & _ranking )`

Cleans up all conditions in this state according to comparison between the corresponding constraints.

**0.15.520.4.98 `solutionSpace()`** `std::vector< smtrat::DoubleInterval > smtrat::vs::State::solutionSpace ( carl::PointerSet< Condition > & _conflictReason ) const`

Determines the solution space of the test candidate of this state regarding to the variable bounds of the father. The solution space consists of one or two disjoint intervals.

## Parameters

|                              |                                                                                                        |
|------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_conflictReason</code> | If the solution space is empty, the conditions being responsible for this conflict are stored in here. |
|------------------------------|--------------------------------------------------------------------------------------------------------|

**Returns**

The disjoint intervals representing the solution space.

**0.15.520.4.99 subResultCombination()** const `SubResultCombination&` `smtrat::vs::State::subResultCombination()` const [inline]

**Returns**

A constant reference to the current combination of conjunction of constraints within the substitution results of this state. Make sure that a substitution result combination exists when calling this method. (using, e.g., `hasSubResultsCombination()`)

**0.15.520.4.100 subResultsSimplified()** bool `smtrat::vs::State::subResultsSimplified()` const [inline]

**Returns**

true, if the substitution results of this state are already simplified.

**0.15.520.4.101 substitution()** const `Substitution&` `smtrat::vs::State::substitution()` const [inline]

**Returns**

A constant reference to the substitution this state considers. Note that this method fails, if this state is the root.

**0.15.520.4.102 substitutionApplicable() [1/2]** bool `smtrat::vs::State::substitutionApplicable()` const  
Checks if a substitution can be applied.

**Returns**

true, if a substitution can be applied. false, else.

**0.15.520.4.103 substitutionApplicable() [2/2]** bool `smtrat::vs::State::substitutionApplicable()` const  
const `smtrat::ConstraintT & _constraint` const  
Checks if the substitution of this state can be applied to the given constraint.

**Parameters**

|                          |                                                                               |
|--------------------------|-------------------------------------------------------------------------------|
| <code>_constraint</code> | The constraint, for which we want to know, if the substitution is applicable. |
|--------------------------|-------------------------------------------------------------------------------|

**Returns**

true, if the substitution can be applied; false, otherwise.

**0.15.520.4.104 substitutionResults()** const `SubstitutionResults&` `smtrat::vs::State::substitutionResults()` const [inline]

**Returns**

A constant reference to the substitution results of this state. Make sure that the substitution results exist when calling this method. (using, e.g., [hasSubstitutionResults\(\)](#))

**0.15.520.4.105 `takeSubResultCombAgain()`** `bool smtrat::vs::State::takeSubResultCombAgain ( )`

`const [inline]`

**Returns**

true, if the current substitution result combination has to be consider once again before considering the next one.

**0.15.520.4.106 `tooHighDegreeConditions()`** `const carl::PointerSet<Condition>& smtrat::vs::vs::State::tooHighDegreeConditions ( ) const [inline]`**Returns**

A constant reference to the conditions of those this state currently considers, which have a too high degree to be tackled of the virtual substitution method.

**0.15.520.4.107 `treeDepth()`** `unsigned smtrat::vs::State::treeDepth ( ) const`**Returns**

The depth of the subtree with this state as root node.

**0.15.520.4.108 `tryToRefreshIndex()`** `bool smtrat::vs::State::tryToRefreshIndex ( ) const [inline]`**Returns**

true, if the index variable of this state shall be recalculated.

**0.15.520.4.109 `type()`** `Type smtrat::vs::State::type ( ) const [inline]`**Returns**

The type of this state: This state has still a substitution to apply or either test candidates shall be generated or a new combination of the substitution results must be found in order to consider it next.

**0.15.520.4.110 `unfinished()`** `bool smtrat::vs::State::unfinished ( ) const [inline]`**Returns**

true, if the currently considered substitution result combination can be extended by taking further substitution results into account.

**0.15.520.4.111 `unfinishedAncestor()`** `bool smtrat::vs::State::unfinishedAncestor ( State *& _unfinAnt )`

Determines (if it exists) a ancestor node, which is unfinished, that is it has still substitution results to consider.

**Parameters**

|                        |                               |
|------------------------|-------------------------------|
| <code>_unfinAnt</code> | The unfinished ancestor node. |
|------------------------|-------------------------------|

**Returns**

true, if it has a unfinished ancestor; false, otherwise.

**0.15.520.4.112 updateBackendCallValuation()** `void smtrat::vs::State::updateBackendCallValuation( )`

Valuates the state's currently considered conditions according to a backend call.

Note: The settings are currently optimized for CAD backend calls.

**0.15.520.4.113 updateIntTestCandidates()** `bool smtrat::vs::State::updateIntTestCandidates( )`

**0.15.520.4.114 updateOcondsOfSubstitutions()** `bool smtrat::vs::State::updateOcondsOfSubstitutions( )`

```
const Substitution & _substitution,
std::vector< State * > & _reactivatedStates)
```

Updates the original conditions of substitutions having the same test candidate as the given.

**Parameters**

|                                 |                                                              |
|---------------------------------|--------------------------------------------------------------|
| <code>_substitution</code>      | The substitution containing the test candidate to check for. |
| <code>_reactivatedStates</code> |                                                              |

**Returns**

true, if the test candidate of the given substitution was already generated; false, otherwise.

**0.15.520.4.115 updateValuation()** `void smtrat::vs::State::updateValuation( bool _lazy )`

Updates the valuation of this state.

**0.15.520.4.116 valuation()** `size_t smtrat::vs::State::valuation( ) const [inline]`

**Returns**

The heuristically determined valuation of this state (see for the description of the corresponding member).

**0.15.520.4.117 variableBounds()** `const VariableBoundsCond& smtrat::vs::State::variableBounds( ) const [inline]`

**Returns**

A constant reference to the object managing and storing the variable's bounds which were extracted from the currently considered conditions of this state.

**0.15.520.4.118 variables()** void smtrat::vs::State::variables ( carl::Variables & *\_variables* ) const  
Finds the variables, which occur in this state.

**Parameters**

|                         |                                          |
|-------------------------|------------------------------------------|
| <code>_variables</code> | The variables which occur in this state. |
|-------------------------|------------------------------------------|

```
0.15.520.4.119 worstConstraintValuation() void smtrat::vs::State::worstConstraintValuation (const std::vector< std::pair< carl::Variable, std::multiset< double >>> & _← varVals)
```

## 0.15.521 **smtrat::StaticUnionFind** Struct Reference

```
#include <UnionFind.h>
```

### Public Types

- using `Value` = `size_t`
- using `Representative` = `Value`
- using `Parents` = `std::vector< Value >`
- using `Ranks` = `std::vector< size_t >`

### Public Member Functions

- void `introduce_variable` (`Value` const &var) noexcept
- auto `find` (`Value` const &val) noexcept -> `Representative`
- void `merge` (`Value` const &a, `Value` const &b) noexcept

### 0.15.521.1 Member Typedef Documentation

**0.15.521.1.1 Parents** using `smtrat::StaticUnionFind::Parents` = `std::vector<Value>`

**0.15.521.1.2 Ranks** using `smtrat::StaticUnionFind::Ranks` = `std::vector<size_t>`

**0.15.521.1.3 Representative** using `smtrat::StaticUnionFind::Representative` = `Value`

**0.15.521.1.4 Value** using `smtrat::StaticUnionFind::Value` = `size_t`

### 0.15.521.2 Member Function Documentation

**0.15.521.2.1 find()** auto smtrat::StaticUnionFind::find ( `Value` const & val ) -> `Representative` [inline], [noexcept]

**0.15.521.2.2 introduce\_variable()** void smtrat::StaticUnionFind::introduce\_variable ( `Value` const & var ) [inline], [noexcept]

**0.15.521.2.3 merge()** void smtrat::StaticUnionFind::merge ( `Value` const & a, `Value` const & b ) [inline], [noexcept]

## 0.15.522 smtrat::statistics::StatisticsSettings Struct Reference

```
#include <StatisticsSettings.h>
```

### Data Fields

- bool `export_as_xml`
- std::string `xml_filename`
- bool `print_as_smllib`

#### 0.15.522.1 Field Documentation

**0.15.522.1.1 `export_as_xml`** bool smtrat::statistics::StatisticsSettings::export\_as\_xml

**0.15.522.1.2 `print_as_smllib`** bool smtrat::statistics::StatisticsSettings::print\_as\_smllib

**0.15.522.1.3 `xml_filename`** std::string smtrat::statistics::StatisticsSettings::xml\_filename

## 0.15.523 smtrat::StrategyGraph Class Reference

```
#include <StrategyGraph.h>
```

### Public Member Functions

- `StrategyGraph()`
- template<typename Module> `BackendLink addBackend` (const std::initializer\_list<`BackendLink`> &backends)
- `BackendLink & addEdge` (std::size\_t from, std::size\_t to)
- `std::size_t addRoot` (const std::initializer\_list<`BackendLink`> &backends)
- bool `hasBranches()` const
- `std::size_t numberOfBranches()` const
- `std::size_t getRoot()` const
- `std::set<std::pair<thread_priority, AbstractModuleFactory*>> getBackends` (std::size\_t vertex, const carl::Condition &condition) const

### Static Public Member Functions

- static bool `TrueCondition` (const carl::Condition &c)

### Friends

- std::ostream & `operator<<` (std::ostream &os, const `StrategyGraph` &sg)

#### 0.15.523.1 Constructor & Destructor Documentation

**0.15.523.1.1 `StrategyGraph()`** smtrat::StrategyGraph::StrategyGraph ( ) [inline]

#### 0.15.523.2 Member Function Documentation

```
0.15.523.2.1 addBackend() template<typename Module >
BackendLink smtrat::StrategyGraph::addBackend (
 const std::initializer_list< BackendLink > & backends) [inline]

0.15.523.2.2 addEdge() BackendLink& smtrat::StrategyGraph::addEdge (
 std::size_t from,
 std::size_t to) [inline]

0.15.523.2.3 addRoot() std::size_t smtrat::StrategyGraph::addRoot (
 const std::initializer_list< BackendLink > & backends) [inline]

0.15.523.2.4 getBackends() std::set<std::pair<thread_priority,AbstractModuleFactory*>> smtrat<-
::StrategyGraph::getBackends (
 std::size_t vertex,
 const carl::Condition & condition) const [inline]

0.15.523.2.5 getRoot() std::size_t smtrat::StrategyGraph::getRoot () const [inline]

0.15.523.2.6 hasBranches() bool smtrat::StrategyGraph::hasBranches () const [inline]

0.15.523.2.7 numberOfBranches() std::size_t smtrat::StrategyGraph::numberOfBranches () const
[inline]

0.15.523.2.8 TrueCondition() static bool smtrat::StrategyGraph::TrueCondition (
 const carl::Condition & c) [inline], [static]
```

### 0.15.523.3 Friends And Related Function Documentation

```
0.15.523.3.1 operator<< std::ostream& operator<< (
 std::ostream & os,
 const StrategyGraph & sg) [friend]
```

## 0.15.524 delta::String Class Reference

This class can be used to create `std::string` objects using streaming operators.  
`#include <utils.h>`

### Public Member Functions

- `template<typename T >`  
`String & operator<< (const T &t)`  
*Add some data to the string.*
- `operator std::string () const`  
*Return data as string.*

### 0.15.524.1 Detailed Description

This class can be used to create `std::string` objects using streaming operators.

### 0.15.524.2 Member Function Documentation

**0.15.524.2.1 operator std::string()** `delta::String::operator std::string ( ) const [inline]`  
 Return data as string.

Returns

Data as string.

**0.15.524.2.2 operator<<()** `template<typename T >`  
`String& delta::String::operator<< (`  
`const T & t ) [inline]`  
 Add some data to the string.

Parameters

|                |       |
|----------------|-------|
| <code>t</code> | Data. |
|----------------|-------|

Returns

`*this`

## 0.15.525 Minisat::StringOption Class Reference

#include <Options.h>

### Public Member Functions

- `StringOption` (const char \*c, const char \*n, const char \*d, const char \*def=NULL)
- `operator const char *` (void) const
- `StringOption & operator=` (const char \*x)
- virtual void `help` (bool verbose=false)

### Static Protected Member Functions

- static `vec< Option * > & getOptionList ()`
- static const char \*& `getUsageString ()`
- static const char \*& `getHelpPrefixString ()`

### Protected Attributes

- const char \* `name`
- const char \* `description`
- const char \* `category`
- const char \* `type_name`

### 0.15.525.1 Constructor & Destructor Documentation

**0.15.525.1.1 StringOption()** `Minisat::StringOption::StringOption (`  
`const char * c,`  
`const char * n,`  
`const char * d,`  
`const char * def = NULL ) [inline]`

### 0.15.525.2 Member Function Documentation

**0.15.525.2.1 `getHelpPrefixString()`** static const char& Minisat::Option::getHelpPrefixString ( ) [inline], [static], [protected], [inherited]

**0.15.525.2.2 `getOptionList()`** static `vec<Option*>&` Minisat::Option::getOptionList ( ) [inline], [static], [protected], [inherited]

**0.15.525.2.3 `getUsageString()`** static const char& Minisat::Option::getUsageString ( ) [inline], [static], [protected], [inherited]

**0.15.525.2.4 `help()`** virtual void Minisat::StringOption::help ( bool verbose = false ) [inline], [virtual]  
Implements [Minisat::Option](#).

**0.15.525.2.5 `operator const char *()`** Minisat::StringOption::operator const char \* ( void ) const [inline]

**0.15.525.2.6 `operator=()`** `StringOption&` Minisat::StringOption::operator= ( const char \* x ) [inline]

### 0.15.525.3 Field Documentation

**0.15.525.3.1 `category`** const char\* Minisat::Option::category [protected], [inherited]

**0.15.525.3.2 `description`** const char\* Minisat::Option::description [protected], [inherited]

**0.15.525.3.3 `name`** const char\* Minisat::Option::name [protected], [inherited]

**0.15.525.3.4 `type_name`** const char\* Minisat::Option::type\_name [protected], [inherited]

## 0.15.526 smrat::parser::StringParser Struct Reference

Parses strings: ".+" with escape sequences \\\" and \\\\\\  
`#include <Lexicon.h>`

### Public Member Functions

- [StringParser \(\)](#)

### Data Fields

- `qi::symbols<char, char> escapes`
- `qi::rule<Iterator, std::string(), Skipper> main`

### 0.15.526.1 Detailed Description

Parses strings: ". +" with escape sequences \\ " and \\\\"

### 0.15.526.2 Constructor & Destructor Documentation

#### 0.15.526.2.1 `StringParser()` smtrat::parser::StringParser::StringParser ( ) [inline]

### 0.15.526.3 Field Documentation

#### 0.15.526.3.1 `escapes` qi::symbols<char, char> smtrat::parser::StringParser::escapes

#### 0.15.526.3.2 `main` qi::rule<Iterator, std::string(), Skipper> smtrat::parser::StringParser::main

## 0.15.527 smtrat::STropModule< Settings > Class Template Reference

```
#include <STropModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }`

### Public Member Functions

- `std::string moduleName () const`
- `STropModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)`
- `bool addCore (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- `void removeCore (ModuleInput::const_iterator _subformula)`

*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- `void updateModel () const`

*Updates the current assignment into the model.*
- `Answer checkCore ()`

*Checks the received formula for consistency.*
- `bool inform (const FormulaT &_constraint)`

*Informs the module about the given constraint.*
- `void deform (const FormulaT &_constraint)`

*The inverse of informing about a constraint.*
- `virtual void init ()`

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
- `bool add (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- `virtual Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`

*Checks the received formula for consistency.*
- `virtual void remove (ModuleInput::const_iterator _subformula)`

*Removes everything related to the given sub-formula of the received formula.*
- `virtual void updateAllModels ()`

- Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
  - unsigned **currentlySatisfiedByBackend** (const **FormulaT** &\_formula) const
  - virtual unsigned **currentlySatisfied** (const **FormulaT** &) const
  - bool **receivedVariable** (carl::Variable::Arg \_var) const
  - **Answer** **solverState** () const
  - std::size\_t **id** () const
  - void **setId** (std::size\_t \_id)  
*Sets this module's unique ID to identify itself.*
  - **thread\_priority** **threadPriority** () const
  - void **setThreadPriority** (**thread\_priority** \_threadPriority)  
*Sets the priority of this module to get a thread for running its check procedure.*
  - const **ModuleInput** \* **pReceivedFormula** () const
  - const **ModuleInput** & **rReceivedFormula** () const
  - const **ModuleInput** \* **pPassedFormula** () const
  - const **ModuleInput** & **rPassedFormula** () const
  - const **Model** & **model** () const
  - const std::vector< **Model** > & **allModels** () const
  - const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
  - const std::vector< **Module** \* > & **usedBackends** () const
  - const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
  - const carl::FastSet< **FormulaT** > & **informedConstraints** () const
  - void **addLemma** (const **FormulaT** &\_lemma, const **LemmaType** &\_lt=**LemmaType::NORMAL**, const **FormulaT** &\_preferredFormula=**FormulaT**(carl::FormulaType::TRUE))  
*Stores a lemma being a valid formula.*
  - bool **hasLemmas** ()  
*Checks whether this module or any of its backends provides any lemmas.*
  - void **clearLemmas** ()  
*Deletes all yet found lemmas.*
  - const std::vector< **Lemma** > & **lemmas** () const
  - **ModuleInput::const\_iterator** **firstUncheckedReceivedSubformula** () const
  - **ModuleInput::const\_iterator** **firstSubformulaToPass** () const
  - void **receivedFormulaChecked** ()  
*Notifies that the received formulas has been checked.*
  - const **smrat::Conditionals** & **answerFound** () const
  - bool **isPreprocessor** () const
  - carl::Variable **objective** () const
  - bool **is\_minimizing** () const
  - void **excludeNotReceivedVariablesFromModel** () const  
*Excludes all variables from the current model, which do not occur in the received formula.*
  - void **updateLemmas** ()  
*Stores all lemmas of any backend of this module in its own lemma vector.*
  - void **collectTheoryPropagations** ()
  - void **collectOrigins** (const **FormulaT** &\_formula, **FormulasT** &\_origins) const  
*Collects the formulas in the given formula, which are part of the received formula.*
  - void **collectOrigins** (const **FormulaT** &\_formula, **FormulaSetT** &\_origins) const
  - bool **isValidInfeasibleSubset** () const
  - void **checkInfSubsetForMinimality** (std::vector< **FormulaSetT** >::const\_iterator \_infssubset, const std::string &\_filename="smaller\_muses", unsigned \_maxSizeDifference=1) const  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
  - virtual std::pair< bool, **FormulaT** > **getReceivedFormulaSimplified** ()

- void `print` (const std::string & \_initiation="\*\*\*") const  
*Prints everything relevant of the solver.*
- void `printReceivedFormula` (const std::string & \_initiation="\*\*\*") const  
*Prints the vector of the received formula.*
- void `printPassedFormula` (const std::string & \_initiation="\*\*\*") const  
*Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const std::string & \_initiation="\*\*\*") const  
*Prints the infeasible subsets.*
- void `printModel` (std::ostream & \_out=std::cout) const  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (std::ostream & \_out=std::cout)  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

### Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` & \_splittingVariable)

### Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< `Branching` > `mLastBranches` = std::vector<`Branching`>(mNumOfBranchVarsToStore, `Branching`(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< `FormulaT` > `mOldSplittingVariables`  
*Reusable splitting variables.*

### Protected Member Functions

- virtual bool `informCore` (const `FormulaT` & \_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` & \_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` & \_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` & \_formula, `FormulasT` & \_origins) const

- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
  - std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
    - std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
  - void `informBackends` (const `FormulaT` &\_constraint)
    - Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
  - Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
  - Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
    - Adds the given formula to the passed formula with no origin.*
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
    - Adds the given formula to the passed formula.*
  - std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
    - Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
  - Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
  - Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
  - void `getInfeasibleSubsets` ()
    - Copies the infeasible subsets of the passed formula.*
  - std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
    - Get the infeasible subsets the given backend provides.*
  - const `Model` & `backendsModel` () const
    - Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
  - void `getBackendsModel` () const
    - Stores all models of a backend in the list of all models of this module.*
  - void `getBackendsAllModels` () const
    - Runs the backend solvers on the passed formula.*
  - virtual `Answer` `runBackends` (bool \_final, bool \_full, carl::Variable \_objective)
    - Runs the backend solvers on the passed formula.*
  - virtual `ModuleInput::iterator` `eraseSubformulaFromPassedFormula` (`ModuleInput::iterator` \_subformula, bool \_ignoreOrigins=false)
    - Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
  - void `clearPassedFormula` ()
    - std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const
      - Merges the two vectors of sets into the first one.*
    - size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
      - bool `probablyLooping` (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const
        - Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
      - bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
        - Branches at the given polynomial and value, using the given premise.*

- Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
  - bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector<`FormulaT`> &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
  - bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector<`FormulaT`> &\_premise=std::vector<`FormulaT`>())
    - void `splitUnequalConstraint` (const `FormulaT` &
 

Adds the following lemmas for the given constraint  $p \neq 0$ .
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
  - Adds a formula to the InformationRelevantFormula.
- const std::set<`FormulaT`> & `getInformationRelevantFormulas` ()
  - Gets all InformationRelevantFormulas.
- bool `isLemmaLevel` (`LemmaLevel` level)
  - Checks if current lemma level is greater or equal to given level.

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
  - Checks whether there is no variable assigned by both models.

## Protected Attributes

- std::vector<`FormulaSetT`> `mInfeasibleSubsets`
  - Stores the infeasible subsets.
- `Manager` \*const `mpManager`
  - A reference to the manager.
- `Model` `mModel`
  - Stores the assignment of the current satisfiable result, if existent.
- std::vector<`Model`> `mAllModels`
  - Stores all satisfying assignments.
- bool `mModelComputed`
  - True, if the model has already been computed.
- bool `mFinalCheck`
  - true, if the check procedure should perform a final check which especially means not to postpone splitting decisions
- bool `mFullCheck`
  - false, if this module should avoid too expensive procedures and rather return unknown instead.
- carl::Variable `mObjectiveVariable`
  - Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.
- std::vector<`TheoryPropagation`> `mTheoryPropagations`
- std::atomic<`Answer`> `mSolverState`
  - States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.
- std::atomic\_bool \* `mBackendsFoundAnswer`
  - This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.
- `Conditionals` `mFoundAnswer`
  - Vector of Booleans: If any of them is true, we have to terminate a running check procedure.
- std::vector<`Module`\*> `mUsedBackends`
  - The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

- `std::vector< Module * > mAllBackends`  
*The backends of this module which have been used.*
- `std::vector< Lemma > mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- `ModuleInput::iterator mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- `carl::FastSet< FormulaT > mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- `carl::FastSet< FormulaT > mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- `unsigned mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- `std::vector< std::size_t > mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.527.1 Member Typedef Documentation

**0.15.527.1.1 SettingsType** `template<typename Settings >`  
`typedef Settings smtrat::STropModule< Settings >::SettingsType`

### 0.15.527.2 Member Enumeration Documentation

**0.15.527.2.1 LemmaType** `enum smtrat::Module::LemmaType : unsigned [strong], [inherited]`

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.527.3 Constructor & Destructor Documentation

**0.15.527.3.1 STropModule()** `template<typename Settings >`  
`smtrat::STropModule< Settings >::STropModule (`  
 `const ModuleInput * _formula,`  
 `Conditionals & _conditionals,`  
 `Manager * _manager = nullptr )`

### 0.15.527.4 Member Function Documentation

**0.15.527.4.1 add()** `bool smtrat::Module::add (`  
 `ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.527.4.2 `addConstraintToInform()`** `void smtrat::Module::addConstraintToInform ( const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.527.4.3 `addCore()`** `template<typename Settings > bool smtrat::STropModule< Settings >::addCore ( ModuleInput::const_iterator _subformula ) [virtual]`

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

False, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; True, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.527.4.4 `addInformationRelevantFormula()`** `void smtrat::Module::addInformationRelevantFormula ( const FormulaT & formula ) [protected], [inherited]`

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.527.4.5 `addLemma()`** `void smtrat::Module::addLemma ( const FormulaT & _lemma, const LemmaType & _lt = LemmaType::NORMAL, const FormulaT & _preferredFormula = FormulaT( carl::FormulaType::TRUE ) ) [inline], [inherited]`

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.527.4.6 addOrigin()** `void smtrat::Module::addOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.527.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator,bool>`

```
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.527.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator,bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.527.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator,bool>`

```
smtrat::Module::addSubformulaToPassedFormula (
```

```
 const FormulaT & _formula,
```

```
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.527.4.10 `addSubformulaToPassedFormula()` [3/3] `std::pair<ModuleInput::iterator,bool>`**

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.527.4.11 `allModels()` `const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]`****Returns**

All satisfying assignments, if existent.

**0.15.527.4.12 `anAnswerFound()` `bool smrat::Module::anAnswerFound () const [inline], [protected], [inherited]`**

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.527.4.13 `answerFound()` `const smrat::Conditionals& smrat::Module::answerFound () const [inline], [inherited]`****Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.527.4.14 `backendsModel()`** const `Model` & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.527.4.15 `branchAt() [1/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.527.4.16 `branchAt() [2/4]`** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.527.4.17 `branchAt() [3/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< `FormulaT` > & \_premise = std::vector<`FormulaT`>() ) [inline], [protected], [inherited]

**0.15.527.4.18 `branchAt() [4/4]`** bool smtrat::Module::branchAt ( const Poly & \_polynomial, bool \_integral, const Rational & \_value, std::vector< `FormulaT` > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [protected], [inherited]

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                            |                                                                                                                                                                                                                |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>   | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>     | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>        | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>      | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code> | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |

**Parameters**

|                                           |                                                                                              |
|-------------------------------------------|----------------------------------------------------------------------------------------------|
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise. |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                |

**0.15.527.4.19 `check()`** Answer smtrat::Module::check (

```
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.527.4.20 `checkCore()`** template<typename Settings >

```
Answer smtrat::STropModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

**Returns**

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; UNKNOWN, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.527.4.21 `checkInfSubsetForMinimality()`** void smtrat::Module::checkInfSubsetForMinimality (

```
 std::vector< FormulaSetT >::const_iterator _infsSubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infsSubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.527.4.22 `checkModel()`** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.527.4.23 `cleanModel()`** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.527.4.24 `clearLemmas()`** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.527.4.25 `clearModel()`** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.527.4.26 `clearModels()`** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.527.4.27 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.527.4.28 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.527.4.29 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.527.4.30 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

**0.15.527.4.31 constraintsToInform()** const carl::FastSet<[FormulaT](#)>& smrat::Module::constraintsToInform( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.527.4.32 currentlySatisfied()** virtual unsigned smrat::Module::currentlySatisfied( const [FormulaT](#) & ) const [inline], [virtual], [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.527.4.33 currentlySatisfiedByBackend()** unsigned smrat::Module::currentlySatisfiedByBackend( const [FormulaT](#) & \_formula ) const [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.527.4.34 deinform()** void smrat::Module::deinform( const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.527.4.35 deinformCore()** virtual void smrat::Module::deinformCore( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.527.4.36 determine\_smallest\_origin()** size\_t smrat::Module::determine\_smallest\_origin( const std::vector< [FormulaT](#) > & origins ) const [protected], [inherited]

## Parameters

|                       |                              |
|-----------------------|------------------------------|
| <code>_origins</code> | A vector of sets of origins. |
|-----------------------|------------------------------|

## Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.527.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`

```
ModuleInput::iterator _subformula,
bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

## Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

## Returns

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.527.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.527.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`

```
const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

## Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

## Returns

**0.15.527.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const [inline], [inherited]`

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.527.4.41 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.527.4.42 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.527.4.43 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.527.4.44 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.527.4.45 getBackendsModel()** `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.527.4.46 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.527.4.47 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (`

`const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.527.4.48 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.527.4.49 getModelEqualities()** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.527.4.50 getOrigins() [1/3]** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulaSetT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.527.4.51 getOrigins() [2/3]** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulasT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.527.4.52 getOrigins() [3/3]** `const FormulaT& smtrat::Module::getOrigins (`

```
ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.527.4.53 getReceivedFormulaSimplified()** `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.527.4.54 hasLemmas()** `bool smrat::Module::hasLemmas () [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.527.4.55 hasValidInfeasibleSubset()** `bool smrat::Module::hasValidInfeasibleSubset () const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.527.4.56 id()** `std::size_t smrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.527.4.57 infeasibleSubsets()** `const std::vector<FormulaSetT>& smrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.527.4.58 inform()** `bool smrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.527.4.59 informBackends()** `void smrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.527.4.60 informCore()** `virtual bool smrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

`false`, if it can be easily decided whether the given constraint is inconsistent; `true`, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

#### 0.15.527.4.61 informedConstraints()

```
const carl::FastSet<FormulaT>& smrat::Module::informed<→
Constraints () const [inline], [inherited]
```

#### Returns

The position of the first constraint of which no backend has been informed about.

#### 0.15.527.4.62 init()

```
void smrat::Module::init () [virtual], [inherited]
```

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

#### 0.15.527.4.63 is\_minimizing()

```
bool smrat::Module::is_minimizing () const [inline], [inherited]
```

#### 0.15.527.4.64 isLemmaLevel()

```
bool smrat::Module::isLemmaLevel (
```

`LemmaLevel level`) [protected], [inherited]

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

#### 0.15.527.4.65 isPreprocessor()

```
bool smrat::Module::isPreprocessor () const [inline], [inherited]
```

#### Returns

`true`, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.527.4.66 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.527.4.67 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & [\\_vecSetA](#), const std::vector< [FormulaT](#) > & [\\_vecSetB](#) ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                          |                                  |
|--------------------------|----------------------------------|
| <a href="#">_vecSetA</a> | A vector of sets of constraints. |
| <a href="#">_vecSetB</a> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.527.4.68 model()** const [Model](#)& smtrat::Module::model ( ) const [inline], [inherited]

**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.527.4.69 modelsDisjoint()** bool smtrat::Module::modelsDisjoint ( const [Model](#) & [\\_modelA](#), const [Model](#) & [\\_modelB](#) ) [static], [protected], [inherited]

Checks whether there is no variable assigned by both models.

**Parameters**

|                         |                                |
|-------------------------|--------------------------------|
| <a href="#">_modelA</a> | The first model to check for.  |
| <a href="#">_modelB</a> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.527.4.70 moduleName()** template<typename Settings > std::string smtrat::STropModule< [Settings](#) >::moduleName ( ) const [inline], [virtual]

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.527.4.71 `objective()`** `carl::Variable smrat::Module::objective ( ) const [inline], [inherited]`

**0.15.527.4.72 `originInReceivedFormula()`** `bool smrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.527.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.527.4.74 `passedFormulaEnd()`** `ModuleInput::iterator smrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.527.4.75 `pPassedFormula()`** `const ModuleInput* smrat::Module::pPassedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.527.4.76 `pReceivedFormula()`** `const ModuleInput* smrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.527.4.77 `print()`** `void smrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.527.4.78 `printAllModels()`** `void smrat::Module::printAllModels ( std::ostream & _out = std::cout ) [inherited]`

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.527.4.79 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the infeasible subsets.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.527.4.80 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]  
Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.527.4.81 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the vector of passed formula.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.527.4.82 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]  
Prints the vector of the received formula.

Parameters

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.527.4.83 probablyLooping()** bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & *\_branchingPolynomial*, const Rational & *\_branchingValue* ) const [protected], [inherited]  
Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

Parameters

|                             |                                                                                     |
|-----------------------------|-------------------------------------------------------------------------------------|
| <i>_branchingPolynomial</i> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <i>_branchingValue</i>      | The value to branch at.                                                             |

**Returns**

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.527.4.84 receivedFormulaChecked()** void smtrat::Module::receivedFormulaChecked () [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.527.4.85 receivedFormulasAsInfeasibleSubset()** void smtrat::Module::receivedFormulasAsInfeasibleSubset (

    ModuleInput::const\_iterator \_subformula ) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

**0.15.527.4.86 receivedVariable()** bool smtrat::Module::receivedVariable (

    carl::Variable::Arg \_var ) const [inline], [inherited]

**0.15.527.4.87 remove()** void smtrat::Module::remove (

    ModuleInput::const\_iterator \_subformula ) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub formula of the received formula to remove. |
|-------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.527.4.88 removeCore()** template<typename Settings >

void smtrat::STropModule< Settings >::removeCore (

    ModuleInput::const\_iterator \_subformula ) [virtual]

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

**Parameters**

|             |                                           |
|-------------|-------------------------------------------|
| _subformula | The position of the subformula to remove. |
|-------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.527.4.89 removeOrigin()** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (

    ModuleInput::iterator \_formula,

    const FormulaT & \_origin ) [inline], [protected], [inherited]

**0.15.527.4.90 removeOrigins()** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (

    ModuleInput::iterator \_formula,

```
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

#### 0.15.527.4.91 **rPassedFormula()** const ModuleInput& smtrat::Module::rPassedFormula ( ) const [inline], [inherited]

##### Returns

A reference to the formula passed to the backends of this module.

#### 0.15.527.4.92 **rReceivedFormula()** const ModuleInput& smtrat::Module::rReceivedFormula ( ) const [inline], [inherited]

##### Returns

A reference to the formula passed to this module.

#### 0.15.527.4.93 **runBackends()** [1/2] virtual Answer smtrat::Module::runBackends ( ) [inline], [protected], [virtual], [inherited] Reimplemented in [smtrat::PModule](#).

#### 0.15.527.4.94 **runBackends()** [2/2] Answer smtrat::Module::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

##### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

##### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

#### 0.15.527.4.95 **setId()** void smtrat::Module::setId (

```
 std::size_t _id) [inline], [inherited]
```

Sets this modules unique ID to identify itself.

##### Parameters

|                                                                                                         |                                    |
|---------------------------------------------------------------------------------------------------------|------------------------------------|
| $\leftrightarrow$<br>$\overleftarrow{\phantom{d}}$<br>$\overrightarrow{\phantom{d}}$<br><code>id</code> | The id to set this module's id to. |
|---------------------------------------------------------------------------------------------------------|------------------------------------|

**0.15.527.4.96 `setThreadPriority()`** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`  
 Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.527.4.97 `solverState()`** `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.527.4.98 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \Leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.527.4.99 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.527.4.100 `updateAllModels()`** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.527.4.101 `updateLemmas()`** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.527.4.102 `updateModel()`** `template<typename Settings >`

`void smtrat::STropModule< Settings >::updateModel ( ) const [virtual]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.527.4.103 usedBackends()** const std::vector<[Module](#)\*>& smrat::Module::usedBackends ( )  
const [inline], [inherited]

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

## 0.15.527.5 Field Documentation

**0.15.527.5.1 mAllBackends** std::vector<[Module](#)\*> smrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

**0.15.527.5.2 mAllModels** std::vector<[Model](#)> smrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.527.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smrat::Module::mBackendsFoundAnswer [protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.527.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.527.5.5 mFinalCheck** bool smrat::Module::mFinalCheck [protected], [inherited]

true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.527.5.6 mFirstPosInLastBranches** std::size\_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.527.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.527.5.8 mFirstUncheckedReceivedSubformula** [ModuleInput](#)::const\_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.527.5.9 mFoundAnswer** [Conditionals](#) smrat::Module::mFoundAnswer [protected], [inherited]

Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.527.5.10 mFullCheck** `bool smtrat::Module::mFullCheck` [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.527.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets` [protected], [inherited]  
Stores the infeasible subsets.

**0.15.527.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints` [protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.527.5.13 mLastBranches** `std::vector< Branching > smtrat::Module::mLastBranches = std::vector<Branching> (mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))` [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.527.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas` [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.527.5.15 mModel** `Model smtrat::Module::mModel` [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.527.5.16 mModelComputed** `bool smtrat::Module::mModelComputed` [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.527.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5` [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.527.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable` [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.527.5.19 mOldSplittingVariables** `std::vector< FormulaT > smtrat::Module::mOldSplittingVariables` [static], [inherited]  
Reusable splitting variables.

**0.15.527.5.20 mpManager** `Manager* const smtrat::Module::mpManager` [protected], [inherited]  
A reference to the manager.

**0.15.527.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter` [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.527.5.22 mSolverState** std::atomic<[Answer](#)> smtrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.527.5.23 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smtrat::Module::mTheory← Propagations [protected], [inherited]**0.15.527.5.24 mUsedBackends** std::vector<[Module\\*](#)> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.527.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.528 benchmax::slurm::SubmitfileProperties Struct Reference

All properties needed to create a submit file.

```
#include <SlurmUtilities.h>
```

### Data Fields

- std::string [file\\_suffix](#)  
*Suffix for job and submit file.*
- std::string [filename\\_jobs](#)  
*Filename of the job list file.*
- std::string [tmp\\_dir](#)  
*Temporary directory for log files.*
- carl::settings::duration [limit\\_time](#)  
*Time limit in seconds.*
- carl::settings::duration [grace\\_time](#)  
*Grace time in seconds.*
- carl::settings::binary\_quantity [limit\\_memory](#)  
*Memory limit in megabytes.*
- std::size\_t [tasks](#)  
*Number of tasks.*
- std::size\_t [slices](#)  
*Number of slices.*

### 0.15.528.1 Detailed Description

All properties needed to create a submit file.

### 0.15.528.2 Field Documentation

**0.15.528.2.1 file\_suffix** std::string benchmax::slurm::SubmitfileProperties::file\_suffix  
Suffix for job and submit file.

**0.15.528.2.2 filename\_jobs** std::string benchmax::slurm::SubmitfileProperties::filename\_jobs  
 Filename of the job list file.

**0.15.528.2.3 grace\_time** carl::settings::duration benchmax::slurm::SubmitfileProperties::grace\_time  
 Grace time in seconds.

**0.15.528.2.4 limit\_memory** carl::settings::binary\_quantity benchmax::slurm::SubmitfileProperties::limit\_memory  
 Memory limit in megabytes.

**0.15.528.2.5 limit\_time** carl::settings::duration benchmax::slurm::SubmitfileProperties::limit\_time  
 Time limit in seconds.

**0.15.528.2.6 slices** std::size\_t benchmax::slurm::SubmitfileProperties::slices  
 Number of slices.

**0.15.528.2.7 tasks** std::size\_t benchmax::slurm::SubmitfileProperties::tasks  
 Number of tasks.

**0.15.528.2.8 tmp\_dir** std::string benchmax::slurm::SubmitfileProperties::tmp\_dir  
 Temporary directory for log files.

## 0.15.529 smrat::vs::Substitution Class Reference

```
#include <Substitution.h>
```

### Public Types

- enum **Type** {
 **NORMAL** , **PLUS\_EPSILON** , **MINUS\_INFINITY** , **PLUS\_INFINITY** ,
 **INVALID** }

*The type of a substitution.*

### Public Member Functions

- Substitution** (const carl::Variable &\_variable, const **Type** &\_type, carl::PointerSet< **Condition** > &&\_oConditions, **smrat::ConstraintsT** &&\_sideCondition=**smrat::ConstraintsT()**)  
*Constructs a substitution with no square root term to map to.*
- Substitution** (const carl::Variable &, const **smrat::SqrtEx** &\_term, const **Type** &\_type, carl::PointerSet< **Condition** > &&\_oConditions, **smrat::ConstraintsT** &&\_sideCondition=**smrat::ConstraintsT()**)  
*Constructs a substitution with a square root term to map to.*
- Substitution** (const **Substitution** &\_substitution)  
*Copy constructor.*
- ~Substitution** ()  
*The destructor.*
- const carl::Variable & **variable** () const
- const **smrat::SqrtEx** & **term** () const

- void `setTerm` (const `smrat::Rational` & `_value`)  
*Sets the substitution term to the given rational.*
- `Type` & `rType` ()
- const `Type` & `type` () const
- `carl::PointerSet< Condition >` & `rOriginalConditions` ()
- const `carl::PointerSet< Condition >` & `originalConditions` () const
- const `smrat::ConstraintsT` & `sideCondition` () const
- const `carl::Variables` & `termVariables` () const
- unsigned `valuate` (bool `_preferMinusInf=true`) const
- bool `operator==` (const `Substitution` &) const
- void `print` (bool `_withOrigins=false`, bool `_withSideConditions=false`, std::ostream & `_out=std::cout`, const std::string & `_init=""`) const  
*Prints this substitution on the given stream, with some additional information.*

### 0.15.529.1 Member Enumeration Documentation

#### 0.15.529.1.1 Type enum `smrat::vs::Substitution::Type`

The type of a substitution.

Enumerator

|                |  |
|----------------|--|
| NORMAL         |  |
| PLUS_EPSILON   |  |
| MINUS_INFINITY |  |
| PLUS_INFINITY  |  |
| INVALID        |  |

### 0.15.529.2 Constructor & Destructor Documentation

#### 0.15.529.2.1 Substitution() [1/3] `smrat::vs::Substitution::Substitution` (

```
 const carl::Variable & _variable,
 const Type & _type,
 carl::PointerSet< Condition > && _oConditions,
 smrat::ConstraintsT && _sideCondition = smrat::ConstraintsT())
```

Constructs a substitution with no square root term to map to.

Parameters

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <code>_variable</code>      | The variable to substitute of the substitution to construct.   |
| <code>_type</code>          | The type of the substitution of the substitution to construct. |
| <code>_oConditions</code>   | The original conditions of the substitution to construct.      |
| <code>_sideCondition</code> | The side conditions of the substitution to construct.          |

#### 0.15.529.2.2 Substitution() [2/3] `smrat::vs::Substitution::Substitution` (

```
 const carl::Variable & _variable,
 const smrat::SqrtEx & _term,
 const Type & _type,
 carl::PointerSet< Condition > && _oConditions,
```

```
smtrat::ConstraintsT && _sideCondition = smtrat::ConstraintsT())
```

Constructs a substitution with a square root term to map to.

#### Parameters

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <code>_variable</code>      | The variable to substitute of the substitution to construct.   |
| <code>_term</code>          | The square root term to which the variable maps to.            |
| <code>_type</code>          | The type of the substitution of the substitution to construct. |
| <code>_oConditions</code>   | The original conditions of the substitution to construct.      |
| <code>_sideCondition</code> | The side conditions of the substitution to construct.          |

**0.15.529.2.3 Substitution()** [3/3] `smtrat::vs::Substitution::Substitution (`  
`const Substitution & _substitution )`

Copy constructor.

**0.15.529.2.4 ~Substitution()** `smtrat::vs::Substitution::~Substitution ( )`  
The destructor.

### 0.15.529.3 Member Function Documentation

**0.15.529.3.1 operator==()** `bool smtrat::vs::Substitution::operator== (`  
`const Substitution & _substitution ) const`

#### Parameters

|                  |                               |
|------------------|-------------------------------|
| <code>The</code> | substitution to compare with. |
|------------------|-------------------------------|

#### Returns

true, if this substitution is equal to the given substitution; false, otherwise.

**0.15.529.3.2 originalConditions()** `const carl::PointerSet<Condition>& smtrat::vs::Substitution::originalConditions ( ) const [inline]`

#### Returns

A constant reference to the original conditions of this substitution.

**0.15.529.3.3 print()** `void smtrat::vs::Substitution::print (`  
`bool _withOrigins = false,`  
`bool _withSideConditions = false,`  
`std::ostream & _out = std::cout,`  
`const std::string & _init = "" ) const`

Prints this substitution on the given stream, with some additional information.

#### Parameters

|                  |                               |
|------------------|-------------------------------|
| <code>The</code> | substitution to compare with. |
|------------------|-------------------------------|

**Returns**

true, if this substitution is less than the given substitution; false, otherwise.

**Parameters**

|                                  |                                                                             |
|----------------------------------|-----------------------------------------------------------------------------|
| <code>_withOrigins</code>        | A flag indicating whether to print also the origins of this substitution.   |
| <code>_withSideConditions</code> | A flag indication whether to also the side conditions of this substitution. |
| <code>_out</code>                | The stream to print on.                                                     |
| <code>_init</code>               | The string to print at the beginning of every row.                          |

**0.15.529.3.4 `rOriginalConditions()`** `carl::PointerSet<Condition>& smtrat::vs::Substitution::rOriginalConditions () [inline]`

**Returns**

A reference to the original conditions of this substitution.

**0.15.529.3.5 `rType()`** `Type& smtrat::vs::Substitution::rType () [inline]`

**Returns**

A reference to the type of this substitution.

**0.15.529.3.6 `setTerm()`** `void smtrat::vs::Substitution::setTerm (const smtrat::Rational & _value) [inline]`

Sets the substitution term to the given rational.

**Parameters**

|                     |                                            |
|---------------------|--------------------------------------------|
| <code>_value</code> | The value to set the substitution term to. |
|---------------------|--------------------------------------------|

**0.15.529.3.7 `sideCondition()`** `const smtrat::ConstraintsT& smtrat::vs::Substitution::sideCondition () const [inline]`

**Returns**

A constant reference to the side condition of this substitution.

**0.15.529.3.8 `term()`** `const smtrat::SqrtEx& smtrat::vs::Substitution::term () const [inline]`

**Returns**

A constant reference to the term this substitution maps its variable to.

**0.15.529.3.9 `termVariables()`** `const carl::Variables& smtrat::vs::Substitution::termVariables () const [inline]`

**Returns**

A constant reference to the variables occurring in the substitution term.

**0.15.529.3.10 type()** const `Type& smtrat::vs::Substitution::type( ) const [inline]`**Returns**

A constant reference to the type of this substitution.

**0.15.529.3.11 valuate()** unsigned smtrat::vs::Substitution::valuate( bool \_preferMinusInf = true ) const**Parameters**

|                              |                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------|
| <code>_preferMinusInf</code> | A flag indicating whether to valuate the substitution type best or otherwise worst. |
|------------------------------|-------------------------------------------------------------------------------------|

**Returns**

The valuation of this substitution according to a heuristic.

**0.15.529.3.12 variable()** const `carl::Variable& smtrat::vs::Substitution::variable( ) const [inline]`**Returns**

A constant reference to the variable this substitution substitutes.

**0.15.530 smtrat::vs::substitutionPointerEqual Struct Reference**

```
#include <Substitution.h>
```

**Public Member Functions**

- `bool operator()(const Substitution *_substitutionA, const Substitution *_substitutionB) const`

**0.15.530.1 Member Function Documentation****0.15.530.1.1 operator()** `bool smtrat::vs::substitutionPointerEqual::operator()( const Substitution * _substitutionA, const Substitution * _substitutionB ) const [inline]`**0.15.531 smtrat::vs::substitutionPointerHash Struct Reference**

```
#include <Substitution.h>
```

**Public Member Functions**

- `size_t operator()(const Substitution *_substitution) const`

**0.15.531.1 Member Function Documentation**

```
0.15.531.1.1 operator() size_t smtrat::vs::substitutionPointerHash::operator() (
 const Substitution * _substitution) const [inline]
```

## 0.15.532 smtrat::cadcells::datastructures::SymbolicInterval Class Reference

A symbolic interal whose bounds are defined by indexed root expressions.

```
#include <roots.h>
```

### Public Member Functions

- [SymbolicInterval \(IndexedRoot root\)](#)  
*Constructs a section.*
- [SymbolicInterval \(Bound lower, Bound upper\)](#)  
*Constructs an interval iwth arbitrary bounds.*
- [SymbolicInterval \(\)](#)  
*Constructs (-oo, oo).*
- [bool is\\_section \(\) const](#)
- [const IndexedRoot & section\\_defining \(\) const](#)  
*In case of a section, the defining indexed root is returned.*
- [const auto & lower \(\) const](#)  
*Return the lower bound.*
- [const auto & upper \(\) const](#)  
*Returns the upper bound.*
- [void polys \(boost::container::flat\\_set< PolyRef > &result\) const](#)
- [boost::container::flat\\_set< PolyRef > polys \(\) const](#)
- [void set\\_to\\_closure \(\)](#)

### 0.15.532.1 Detailed Description

A symbolic interal whose bounds are defined by indexed root expressions.

Bounds can be infty, strict or weak. A special case is a section where the lower and upper bound are equal and weak.

### 0.15.532.2 Constructor & Destructor Documentation

```
0.15.532.2.1 SymbolicInterval() [1/3] smtrat::cadcells::datastructures::SymbolicInterval::←
SymbolicInterval (
 IndexedRoot root) [inline]
```

Constructs a section.

```
0.15.532.2.2 SymbolicInterval() [2/3] smtrat::cadcells::datastructures::SymbolicInterval::←
SymbolicInterval (
 Bound lower,
 Bound upper) [inline]
```

Constructs an interval iwth arbitrary bounds.

```
0.15.532.2.3 SymbolicInterval() [3/3] smtrat::cadcells::datastructures::SymbolicInterval::←
SymbolicInterval () [inline]
```

Constructs (-oo, oo).

### 0.15.532.3 Member Function Documentation

**0.15.532.3.1 `is_section()`** `bool smtrat::cadcells::datastructures::SymbolicInterval::is_section ( ) const [inline]`

**0.15.532.3.2 `lower()`** `const auto& smtrat::cadcells::datastructures::SymbolicInterval::lower ( ) const [inline]`  
Return the lower bound.

**0.15.532.3.3 `polys() [1/2]`** `boost::container::flat_set<PolyRef> smtrat::cadcells::datastructures::SymbolicInterval::polys ( ) const [inline]`

**0.15.532.3.4 `polys() [2/2]`** `void smtrat::cadcells::datastructures::SymbolicInterval::polys ( boost::container::flat_set< PolyRef > & result ) const [inline]`

**0.15.532.3.5 `section_defining()`** `const IndexedRoot& smtrat::cadcells::datastructures::SymbolicInterval::section_defining ( ) const [inline]`  
In case of a section, the defining indexed root is returned.

**0.15.532.3.6 `set_to_closure()`** `void smtrat::cadcells::datastructures::SymbolicInterval::set_to_closure ( ) [inline]`

**0.15.532.3.7 `upper()`** `const auto& smtrat::cadcells::datastructures::SymbolicInterval::upper ( ) const [inline]`  
Returns the upper bound.

## 0.15.533 `delta::SymbolParser` Struct Reference

This class is a `boost::spirit::qi` grammar that matches all kinds of smtlib symbols, numbers etc.  
`#include <Parser.h>`

### Public Member Functions

- [SymbolParser \(\)](#)

### Data Fields

- `qi::rule< Iterator, std::string(), Skipper > main`
- `qi::rule< Iterator, std::string(), Skipper > quoted`
- `qi::rule< Iterator, std::string(), Skipper > simple`

### 0.15.533.1 Detailed Description

This class is a `boost::spirit::qi` grammar that matches all kinds of smtlib symbols, numbers etc.

### 0.15.533.2 Constructor & Destructor Documentation

**0.15.533.2.1 `SymbolParser()`** `delta::SymbolParser::SymbolParser ( ) [inline]`

### 0.15.533.3 Field Documentation

**0.15.533.3.1 main** `qi::rule<Iterator, std::string(), Skipper> delta::SymbolParser::main`

**0.15.533.3.2 quoted** `qi::rule<Iterator, std::string(), Skipper> delta::SymbolParser::quoted`

**0.15.533.3.3 simple** `qi::rule<Iterator, std::string(), Skipper> delta::SymbolParser::simple`

## 0.15.534 smtrat::parser::SymbolParser Struct Reference

```
#include <Lexicon.h>
```

### Public Member Functions

- `SymbolParser ()`

### Data Fields

- `qi::rule< Iterator, std::string(), Skipper > main`
- `qi::rule< Iterator, std::string(), Skipper > quoted`
- `SimpleSymbolParser simple`

### 0.15.534.1 Constructor & Destructor Documentation

**0.15.534.1.1 SymbolParser()** `smtrat::parser::SymbolParser::SymbolParser () [inline]`

### 0.15.534.2 Field Documentation

**0.15.534.2.1 main** `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::SymbolParser::main`

**0.15.534.2.2 quoted** `qi::rule<Iterator, std::string(), Skipper> smtrat::parser::SymbolParser::quoted`

**0.15.534.2.3 simple** `SimpleSymbolParser smtrat::parser::SymbolParser::simple`

## 0.15.535 smtrat::SymmetryModule< Settings > Class Template Reference

```
#include <SymmetryModule.h>
```

### Public Types

- `typedef Settings SettingsType`
- `enum class LemmaType : unsigned { NORMAL = 0, PERMANENT = 1 }`

## Public Member Functions

- `SymmetryModule (const ModuleInput *_formula, Conditionals &_conditionals, Manager *_manager=nullptr)`  
 Checks the received formula for consistency.
- `bool isPreprocessor () const`
- `bool appliedPreprocessing () const`
- `bool add (ModuleInput::const_iterator _subformula)`  
 The module has to take the given sub-formula of the received formula into account.
- `void remove (ModuleInput::const_iterator _subformula)`  
 Removes everything related to the given sub-formula of the received formula.
- `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`  
 Checks the received formula for consistency.
- `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
 Runs the backend solvers on the passed formula.
- `virtual Answer runBackends ()`
- `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void updateModel () const`  
 Updates the current assignment into the model.
- `bool inform (const FormulaT &_constraint)`  
 Informs the module about the given constraint.
- `void deinform (const FormulaT &_constraint)`  
 The inverse of informing about a constraint.
- `virtual void init ()`  
 Informs all backends about the so far encountered constraints, which have not yet been communicated.
- `virtual void updateAllModels ()`  
 Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.
- `virtual std::list< std::vector< carl::Variable > > getModelEqualities () const`  
 Partition the variables from the current model into equivalence classes according to their assigned value.
  - `unsigned currentlySatisfiedByBackend (const FormulaT &_formula) const`
  - `virtual unsigned currentlySatisfied (const FormulaT &) const`
  - `bool receivedVariable (carl::Variable::Arg _var) const`
  - `Answer solverState () const`
  - `std::size_t id () const`
  - `void setId (std::size_t _id)`  
 Sets this modules unique ID to identify itself.
  - `thread_priority threadPriority () const`
  - `void setThreadPriority (thread_priority _threadPriority)`  
 Sets the priority of this module to get a thread for running its check procedure.
  - `const ModuleInput * pReceivedFormula () const`
  - `const ModuleInput & rReceivedFormula () const`
  - `const ModuleInput * pPassedFormula () const`
  - `const ModuleInput & rPassedFormula () const`
  - `const Model & model () const`
  - `const std::vector< Model > & allModels () const`
  - `const std::vector< FormulaSetT > & infeasibleSubsets () const`
  - `const std::vector< Module * > & usedBackends () const`
  - `const carl::FastSet< FormulaT > & constraintsToInform () const`
  - `const carl::FastSet< FormulaT > & informedConstraints () const`
  - `void addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`  
 Stores a lemma being a valid formula.

- `bool hasLemmas ()`  
*Checks whether this module or any of its backends provides any lemmas.*
- `void clearLemmas ()`  
*Deletes all yet found lemmas.*
- `const std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- `void receivedFormulaChecked ()`  
*Notifies that the received formulas has been checked.*
- `const smrat::Conditionals & answerFound () const`
- `virtual std::string moduleName () const`
- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`  
*Excludes all variables from the current model, which do not occur in the received formula.*
- `void updateLemmas ()`  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
*Collects the formulas in the given formula, which are part of the received formula.*
- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- `void print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- `void printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- `void printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- `void printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*
- `void printModel (std::ostream &_out=std::cout) const`  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- `void printAllModels (std::ostream &_out=std::cout)`  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- `static void freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- `static size_t mNumOfBranchVarsToStore = 5`  
*The number of different variables to consider for a probable infinite loop of branchings.*
- `static std::vector< Branching > mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`  
*Stores the last branches in a cycle buffer.*
- `static size_t mFirstPosInLastBranches = 0`  
*The beginning of the cyclic buffer storing the last branches.*
- `static std::vector< FormulaT > mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` &\_constraint)
 

*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` &\_constraint)
 

*The inverse of informing about a constraint.*
- virtual bool `addCore` (`ModuleInput::const_iterator` formula)
 

*The module has to take the given sub-formula of the received formula into account.*
- virtual void `removeCore` (`ModuleInput::const_iterator` formula)
 

*Removes everything related to the given sub-formula of the received formula.*
- bool `anAnswerFound` () const
 

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const
 

*Clears the assignment, if any was found.*
- void `clearModels` () const
 

*Clears all assignments, if any was found.*
- void `cleanModel` () const
 

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator` `passedFormulaBegin` ()
- `ModuleInput::iterator` `passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
 

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const
 

*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()

- Copies the infeasible subsets of the passed formula.
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
 

Get the infeasible subsets the given backend provides.
- const `Model` & `backendsModel` () const
 

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.
- void `getBackendsModel` () const
- void `getBackendsAllModels` () const
 

Stores all models of a backend in the list of all models of this module.
- virtual `ModuleInput`::iterator `eraseSubformulaFromPassedFormula` (`ModuleInput`::iterator \_subformula, bool \_ignoreOrigins=false)
 

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.
- void `clearPassedFormula` ()
- std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const
 

Merges the two vectors of sets into the first one.
- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename `Poly`::`PolyType` &\_branchingPolynomial, const `Rational` &\_branchingValue) const
 

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (const `Poly` &\_polynomial, bool \_integral, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &&\_premise=std::vector< `FormulaT` >())
 

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- bool `branchAt` (carl::Variable::Arg \_var, const `Rational` &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
 

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.
- void `splitUnequalConstraint` (const `FormulaT` &)
 

Adds the following lemmas for the given constraint  $p \neq 0$ .
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const `FormulaT` &formula)
 

Adds a formula to the InformationRelevantFormula.
- const std::set< `FormulaT` > & `getInformationRelevantFormulas` ()
 

Gets all InformationRelevantFormulas.
- bool `isLemmaLevel` (`LemmaLevel` level)
 

Checks if current lemma level is greater or equal to given level.

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)
 

Checks whether there is no variable assigned by both models.

## Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`

Stores the infeasible subsets.
- `Manager` \*const `mpManager`

A reference to the manager.
- `Model` `mModel`

- std::vector< Model > mAllModels
 

*Stores the assignment of the current satisfiable result, if existent.*
- bool mModelComputed
 

*Stores all satisfying assignments.*
- bool mFinalCheck
 

*True, if the model has already been computed.*
- bool mFullCheck
 

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool mFullCheck
 

*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable mObjectiveVariable
 

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< TheoryPropagation > mTheoryPropagations
- std::atomic< Answer > mSolverState
 

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* mBackendsFoundAnswer
 

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals mFoundAnswer
 

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< Module \* > mUsedBackends
 

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< Module \* > mAllBackends
 

*The backends of this module which have been used.*
- std::vector< Lemma > mLemmas
 

*Stores the lemmas being valid formulas this module or its backends made.*
- ModuleInput::iterator mFirstSubformulaToPass
 

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< FormulaT > mConstraintsToInform
 

*Stores the constraints which the backends must be informed about.*
- carl::FastSet< FormulaT > mInformedConstraints
 

*Stores the position of the first constraint of which no backend has been informed about.*
- ModuleInput::const\_iterator mFirstUncheckedReceivedSubformula
 

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned mSmallerMusesCheckCounter
 

*Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > mVariableCounters
 

*Maps variables to the number of their occurrences.*

### 0.15.535.1 Member Typedef Documentation

```
0.15.535.1.1 SettingsType template<typename Settings >
typedef Settings smtrat::SymmetryModule< Settings >::SettingsType
```

### 0.15.535.2 Member Enumeration Documentation

```
0.15.535.2.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

**Enumerator**

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

**0.15.535.3 Constructor & Destructor Documentation****0.15.535.3.1 SymmetryModule()** `template<typename Settings >`

```
smtrat::SymmetryModule< Settings >::SymmetryModule (
 const ModuleInput * _formula,
 Conditionals & _conditionals,
 Manager * _manager = nullptr)
```

**0.15.535.3.2 ~SymmetryModule()** `template<typename Settings >`

```
smtrat::SymmetryModule< Settings >::~SymmetryModule ()
```

**0.15.535.4 Member Function Documentation****0.15.535.4.1 add()** `bool smtrat::PModule::add (`

```
 ModuleInput::const_iterator _subformula) [inherited]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.535.4.2 addConstraintToInform()** `void smtrat::Module::addConstraintToInform (`

```
 const FormulaT & _constraint) [protected], [virtual], [inherited]
```

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in `smtrat::SATModule< Settings >`.

**0.15.535.4.3 addCore()** `virtual bool smtrat::Module::addCore (`

```
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::SATModule< Settings >](#), [smrat::PFEModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICEModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::GBModule< Settings >](#).

**0.15.535.4.4 addInformationRelevantFormula()** void smrat::Module::addInformationRelevantFormula (

const [FormulaT](#) & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.535.4.5 addLemma()** void smrat::Module::addLemma (

const [FormulaT](#) & \_lemma,

const [LemmaType](#) & \_lt = [LemmaType::NORMAL](#),

const [FormulaT](#) & \_preferredFormula = [FormulaT\( carl::FormulaType::TRUE \)](#) ) [inline],

[inherited]

Stores a lemma being a valid formula.

**Parameters**

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.535.4.6 addOrigin()** void smrat::Module::addOrigin (

[ModuleInput::iterator](#) \_formula,

const [FormulaT](#) & \_origin ) [inline], [protected], [inherited]

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

**0.15.535.4.7 addReceivedSubformulaToPassedFormula()** std::pair<[ModuleInput::iterator](#), bool> smrat::Module::addReceivedSubformulaToPassedFormula (

---

```
ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.535.4.8 addSubformulaToPassedFormula()** [1/3] `std::pair<ModuleInput::iterator,bool> smtrat<->`  
`::Module::addSubformulaToPassedFormula (`  
 `const FormulaT & _formula ) [inline], [protected], [inherited]`

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.535.4.9 addSubformulaToPassedFormula()** [2/3] `std::pair<ModuleInput::iterator,bool> smtrat<->`  
`::Module::addSubformulaToPassedFormula (`  
 `const FormulaT & _formula,`  
 `const FormulaT & _origin ) [inline], [protected], [inherited]`

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.535.4.10 addSubformulaToPassedFormula()** [3/3] `std::pair<ModuleInput::iterator,bool>`  
`smtrat::Module::addSubformulaToPassedFormula (`  
 `const FormulaT & _formula,`  
 `const std::shared_ptr< std::vector< FormulaT >> & _origins ) [inline], [protected],`  
`[inherited]`

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.535.4.11 `allModels()`** `const std::vector<Model>& smtrat::Module::allModels () const [inline], [inherited]`

**Returns**

All satisfying assignments, if existent.

**0.15.535.4.12 `anAnswerFound()`** `bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.535.4.13 `answerFound()`** `const smtrat::Conditionals& smtrat::Module::answerFound () const [inline], [inherited]`

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.535.4.14 `appliedPreprocessing()`** `bool smtrat::PModule::appliedPreprocessing () const [inline], [inherited]`

**Returns**

true, if the current received formula has been simplified and the result of this simplification is stored in the passed formula.

**0.15.535.4.15 `backendsModel()`** `const Model & smtrat::Module::backendsModel () const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

---

**0.15.535.4.16 branchAt() [1/4]** bool smrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.535.4.17 branchAt() [2/4]** bool smrat::Module::branchAt (

```
carl::Variable::Arg _var,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

**0.15.535.4.18 branchAt() [3/4]** bool smrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false,
const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

**0.15.535.4.19 branchAt() [4/4]** bool smrat::Module::branchAt (

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.535.4.20 check() Answer smrat::PModule::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

```
0.15.535.4.21 checkCore() template<typename Settings >
Answer smrat::SymmetryModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

```
0.15.535.4.22 checkInfeasibleSubsetForMinimality() void smrat::Module::checkInfeasibleSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.535.4.23 checkModel() unsigned smrat::Module::checkModel () const [protected], [inherited]
```

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.535.4.24 cleanModel()** void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.535.4.25 clearLemmas()** void smtrat::Module::clearLemmas ( ) [inline], [inherited]

Deletes all yet found lemmas.

**0.15.535.4.26 clearModel()** void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]

Clears the assignment, if any was found.

**0.15.535.4.27 clearModels()** void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]

Clears all assignments, if any was found.

**0.15.535.4.28 clearPassedFormula()** void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]

**0.15.535.4.29 collectOrigins() [1/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulaSetT` & `_origins` ) const [inherited]

**0.15.535.4.30 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const `FormulaT` & `_formula`, `FormulasT` & `_origins` ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.535.4.31 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.535.4.32 constraintsToInform()** const carl::FastSet<`FormulaT`>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.535.4.33 currentlySatisfied()** `virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.535.4.34 currentlySatisfiedByBackend()** `unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & _formula ) const [inherited]`

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.535.4.35 deinform()** `void smtrat::Module::deinform ( const FormulaT & _constraint ) [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.535.4.36 deinformCore()** `virtual void smtrat::Module::deinformCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.535.4.37 determine\_smallest\_origin()** `size_t smtrat::Module::determine_smallest_origin ( const std::vector< FormulaT > & origins ) const [protected], [inherited]`

**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.535.4.38 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula ( ModuleInput::iterator _subformula, bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.535.4.39 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const` [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.535.4.40 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin ( const std::vector< FormulaT > & _origins ) const` [protected], [inherited]

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

**Returns**

**0.15.535.4.41 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass ( ) const` [inline], [inherited]

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.535.4.42 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const` [inline], [inherited]

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.535.4.43 freeSplittingVariable()** static void smtrat::Module::freeSplittingVariable ( const `FormulaT` & `_splittingVariable` ) [inline], [static], [inherited]

**0.15.535.4.44 generateTrivialInfeasibleSubset()** void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]  
Stores the trivial infeasible subset being the set of received formulas.

**0.15.535.4.45 getBackendsAllModels()** void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]  
Stores all models of a backend in the list of all models of this module.

**0.15.535.4.46 getBackendsModel()** void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]

**0.15.535.4.47 getInfeasibleSubsets() [1/2]** void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]  
Copies the infeasible subsets of the passed formula.

**0.15.535.4.48 getInfeasibleSubsets() [2/2]** std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets ( const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.  
Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

#### Parameters

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

#### Returns

The infeasible subsets the given backend provides.

**0.15.535.4.49 getInformationRelevantFormulas()** const std::set< `FormulaT` > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]  
Gets all InformationRelevantFormulas.

#### Returns

Set of all formulas

**0.15.535.4.50 getModelEqualities()** std::list< std::vector< `carl::Variable` > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]

Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.535.4.51 getOrigins() [1/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulaSetT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.535.4.52 getOrigins() [2/3]** void smtrat::Module::getOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inline], [protected], [inherited]

**Parameters**

|          |  |
|----------|--|
| _formula |  |
| _origins |  |

**0.15.535.4.53 getOrigins() [3/3]** const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const\_iterator \_formula ) const [inline], [protected], [inherited]

Gets the origins of the passed formula at the given position.

**Parameters**

|          |                                                   |
|----------|---------------------------------------------------|
| _formula | The position of a formula in the passed formulas. |
|----------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.535.4.54 getReceivedFormulaSimplified()** std::pair< bool, FormulaT > smtrat::PModule::getReceivedFormulaSimplified ( ) [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented from [smtrat::Module](#).

**0.15.535.4.55 hasLemmas()** bool smtrat::Module::hasLemmas ( ) [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.535.4.56 hasValidInfeasibleSubset()** bool smtrat::Module::hasValidInfeasibleSubset ( ) const [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.535.4.57 `id()`** `std::size_t smtrat::Module::id ( ) const [inline], [inherited]`

#### Returns

A unique ID to identify this module instance.

**0.15.535.4.58 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets ( ) const [inline], [inherited]`

#### Returns

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.535.4.59 `inform()`** `bool smtrat::Module::inform ( const FormulaT & _constraint ) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.535.4.60 `informBackends()`** `void smtrat::Module::informBackends ( const FormulaT & _constraint ) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.535.4.61 `informCore()`** `virtual bool smtrat::Module::informCore ( const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smtrat::UnionFindModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#),

`smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, smrat::BVMModule< Settings >, smrat::ICPModule< Settings >, smrat::ICPModule< ICPSettings4 >, and smrat::ICPModule< ICPSettings1 >.`

**0.15.535.4.62 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.535.4.63 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in `smrat::PBPPModule< Settings >, smrat::PBGaussModule< Settings >, smrat::NRAILModule< Settings >, smrat::NewCoveringModule< Settings >, smrat::NewCADModule< Settings >, smrat::LRAModule< Settings >, smrat::LRAModule< LRASettingsICP >, smrat::LRAModule< LRASettings1 >, smrat::IntBlastModule< Settings >, smrat::FPPModule< Settings >, smrat::EQModule< Settings >, smrat::CurryModule< Settings >, and smrat::BVMModule< Settings >.`

**0.15.535.4.64 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.535.4.65 isLemmaLevel()** `bool smrat::Module::isLemmaLevel( LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.535.4.66 isPreprocessor()** `bool smrat::PModule::isPreprocessor( ) const [inline], [inherited]`

**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with `getReceivedFormulaSimplified()`.

**0.15.535.4.67 lemmas()** `const std::vector<Lemma>& smrat::Module::lemmas( ) const [inline], [inherited]`

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.535.4.68 merge()** `std::vector< FormulaT > smrat::Module::merge( const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.535.4.69 `model()`** `const Model& smrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.535.4.70 `modelsDisjoint()`** `bool smrat::Module::modelsDisjoint (`

```
const Model & _modelA,
const Model & _modelB) [static], [protected], [inherited]
```

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.535.4.71 `moduleName()`** `virtual std::string smrat::Module::moduleName () const [inline], [virtual], [inherited]`**Returns**

The name of the given module type as name.

Reimplemented in [smrat::VSModule< Settings >](#), [smrat::UnionFindModule< Settings >](#), [smrat::UFCegarModule< Settings >](#), [smrat::STropModule< Settings >](#), [smrat::SplitSOSModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LVEModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntEqModule< Settings >](#), [smrat::IntBlastModule< Settings >](#), [smrat::IncWidthModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), [smrat::ICPModule< ICPSettings1 >](#), [smrat::GBModule< Settings >](#), [smrat::FouMoModule< Settings >](#), [smrat::EQPreprocessingModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::CubeLIAModule< Settings >](#), [smrat::CSplitModule< Settings >](#), [smrat::BVMModule< Settings >](#), and [smrat::BEModule< Settings >](#).

**0.15.535.4.72 `objective()`** `carl::Variable smrat::Module::objective () const [inline], [inherited]`**0.15.535.4.73 `originInReceivedFormula()`** `bool smrat::Module::originInReceivedFormula (`

```
const FormulaT & _origin) const [protected], [inherited]
```

**0.15.535.4.74 passedFormulaBegin()** `ModuleInput::iterator smtrat::Module::passedFormulaBegin ( )` [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.535.4.75 passedFormulaEnd()** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )` [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.535.4.76 pPassedFormula()** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const` [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.535.4.77 pReceivedFormula()** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const` [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.535.4.78 print()** `void smtrat::Module::print ( const std::string & _initiation = "***" ) const` [inherited]

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.535.4.79 printAllModels()** `void smtrat::Module::printAllModels ( std::ostream & _out = std::cout )` [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.535.4.80 printInfeasibleSubsets()** `void smtrat::Module::printInfeasibleSubsets ( const std::string & _initiation = "***" ) const` [inherited]

Prints the infeasible subsets.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.535.4.81 `printModel()`** `void smtrat::Module::printModel ( std::ostream & _out = std::cout ) const [inherited]`

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.535.4.82 `printPassedFormula()`** `void smtrat::Module::printPassedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of passed formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.535.4.83 `printReceivedFormula()`** `void smtrat::Module::printReceivedFormula ( const std::string & _initiation = "***" ) const [inherited]`

Prints the vector of the received formula.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.535.4.84 `probablyLooping()`** `bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & _branchingPolynomial, const Rational & _branchingValue ) const [protected], [inherited]`

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

**Parameters**

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

**Returns**

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.535.4.85 `receivedFormulaChecked()`** `void smtrat::Module::receivedFormulaChecked ( ) [inline], [inherited]`

Notifies that the received formulas has been checked.

**0.15.535.4.86 `receivedFormulasAsInfeasibleSubset()`** void smtrat::Module::receivedFormulasAsInfeasibleSubset (

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

**0.15.535.4.87 `receivedVariable()`** bool smtrat::Module::receivedVariable (

```
 carl::Variable::Arg _var) const [inline], [inherited]
```

**0.15.535.4.88 `remove()`** void smtrat::PModule::remove (

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.535.4.89 `removeCore()`** virtual void smtrat::Module::removeCore (

```
 ModuleInput::const_iterator formula) [inline], [protected], [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub formula of the received formula to remove. |
|----------------------|----------------------------------------------------|

Reimplemented in [smtrat::VSModule< Settings >](#), [smtrat::SATModule< Settings >](#), [smtrat::PFEModule< Settings >](#), [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), [smtrat::ICPModule< ICPSettings1 >](#), [smtrat::UnionFindModule< Settings >](#), [smtrat::UFCegarModule< Settings >](#), [smtrat::STropModule< Settings >](#), [smtrat::PBPPModule< Settings >](#), [smtrat::PBGaussModule< Settings >](#), [smtrat::NRAILModule< Settings >](#), [smtrat::NewCoveringModule< Settings >](#), [smtrat::NewCADModule< Settings >](#), [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), [smtrat::LRAModule< LRASettings1 >](#), [smtrat::IntEqModule< Settings >](#), [smtrat::IntBlastModule< Settings >](#), [smtrat::IncWidthModule< Settings >](#), [smtrat::ICEModule< Settings >](#), [smtrat::FPPModule< Settings >](#), [smtrat::FouMoModule< Settings >](#), [smtrat::EQModule< Settings >](#), [smtrat::CurryModule< Settings >](#), [smtrat::CubeLIAModule< Settings >](#), [smtrat::CSplitModule< Settings >](#), [smtrat::BVMModule< Settings >](#), and [smtrat::GBModule< Settings >](#).

**0.15.535.4.90 `removeOrigin()`** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.535.4.91 `removeOrigins()`** std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

#### 0.15.535.4.92 rPassedFormula() const ModuleInput& smtrat::Module::rPassedFormula () const [inline], [inherited]

##### Returns

A reference to the formula passed to the backends of this module.

#### 0.15.535.4.93 rReceivedFormula() const ModuleInput& smtrat::Module::rReceivedFormula () const [inline], [inherited]

##### Returns

A reference to the formula passed to this module.

#### 0.15.535.4.94 runBackends() [1/2] virtual Answer smtrat::PModule::runBackends () [inline], [virtual], [inherited]

Reimplemented from [smtrat::Module](#).

#### 0.15.535.4.95 runBackends() [2/2] Answer smtrat::PModule::runBackends ( bool \_final, bool \_full, carl::Variable \_objective ) [virtual], [inherited]

Runs the backend solvers on the passed formula.

##### Parameters

|            |                                                                                                                                       |
|------------|---------------------------------------------------------------------------------------------------------------------------------------|
| _final     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| _full      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| _objective | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

##### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

#### 0.15.535.4.96 setId() void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]

Sets this modules unique ID to identify itself.

##### Parameters

|                     |                                    |
|---------------------|------------------------------------|
| ↔<br>↔<br><i>id</i> | The id to set this module's id to. |
|---------------------|------------------------------------|

**0.15.535.4.97 `setThreadPriority()`** `void smtrat::Module::setThreadPriority ( thread_priority _threadPriority ) [inline], [inherited]`  
Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.535.4.98 `solverState()`** `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.535.4.99 `splitUnequalConstraint()`** `void smtrat::Module::splitUnequalConstraint ( const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\text{not}(p < 0 \text{ and } p > 0)$

#### Parameters

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.535.4.100 `threadPriority()`** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.535.4.101 `updateAllModels()`** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.535.4.102 `updateLemmas()`** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.535.4.103 `updateModel()`** `void smtrat::PModule::updateModel ( ) const [virtual], [inherited]`

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.

Reimplemented from [smtrat::Module](#).

**0.15.535.4.104 `usedBackends()`** `const std::vector<Module*>& smtrat::Module::usedBackends ( ) const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.535.5 Field Documentation**

**0.15.535.5.1 mAllBackends** `std::vector<Module*> smrat::Module::mAllBackends [protected], [inherited]`

The backends of this module which have been used.

**0.15.535.5.2 mAllModels** `std::vector<Model> smrat::Module::mAllModels [mutable], [protected], [inherited]`

Stores all satisfying assignments.

**0.15.535.5.3 mBackendsFoundAnswer** `std::atomic_bool* smrat::Module::mBackendsFoundAnswer [protected], [inherited]`

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.535.5.4 mConstraintsToInform** `carl::FastSet<FormulaT> smrat::Module::mConstraintsToInform [protected], [inherited]`

Stores the constraints which the backends must be informed about.

**0.15.535.5.5 mFinalCheck** `bool smrat::Module::mFinalCheck [protected], [inherited]`  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.535.5.6 mFirstPosInLastBranches** `std::size_t smrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]`

The beginning of the cyclic buffer storing the last branches.

**0.15.535.5.7 mFirstSubformulaToPass** `ModuleInput::iterator smrat::Module::mFirstSubformulaToPass [protected], [inherited]`

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.535.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.535.5.9 mFoundAnswer** `Conditionals smrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.535.5.10 mFullCheck** `bool smrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.535.5.11 mInfeasibleSubsets** std::vector<[FormulaSetT](#)> smtrat::Module::mInfeasibleSubsets  
[protected], [inherited]  
Stores the infeasible subsets.

**0.15.535.5.12 mInformedConstraints** carl::FastSet<[FormulaT](#)> smtrat::Module::mInformedConstraints  
[protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.535.5.13 mLastBranches** std::vector< [Branching](#) > smtrat::Module::mLastBranches = std::vector<[Branching](#)>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static],  
[inherited]  
Stores the last branches in a cycle buffer.

**0.15.535.5.14 mLemmas** std::vector<[Lemma](#)> smtrat::Module::mLemmas [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.535.5.15 mModel** [Model](#) smtrat::Module::mModel [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.535.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected],  
[inherited]  
True, if the model has already been computed.

**0.15.535.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.535.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected],  
[inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.535.5.19 mOldSplittingVariables** std::vector< [FormulaT](#) > smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.535.5.20 mpManager** [Manager](#)\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.535.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.535.5.22 mSolverState** std::atomic<[Answer](#)> smtrat::Module::mSolverState [protected], [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.535.5.23 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smtrat::Module::mTheory← Propagations [protected], [inherited]

**0.15.535.5.24 mUsedBackends** std::vector<[Module](#)\*> smtrat::Module::mUsedBackends [protected], [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.535.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.536 smtrat::lra::Tableau< Settings, T1, T2 > Class Template Reference

```
#include <Tableau.h>
```

### Data Structures

- struct [LearnedBound](#)

### Public Member Functions

- [Tableau \(ModuleInput::iterator \\_defaultBoundPosition\)](#)
- [~Tableau \(\)](#)
- void [setSize \(size\\_t \\_expectedHeight, size\\_t \\_expectedWidth, size\\_t \\_expectedNumberOfBounds\)](#)
- size\_t [size \(\) const](#)
- void [setBlandsRuleStart \(size\\_t \\_start\)](#)
- const std::vector< [Variable< T1, T2 > \\*rows \(\) const](#)
- const std::vector< [Variable< T1, T2 > \\*columns \(\) const](#)
- const carl::FastMap< carl::Variable, [Variable< T1, T2 > \\*originalVars \(\) const](#)
- const carl::FastPointerMap< typename Poly::PolyType, [Variable< T1, T2 > \\*slackVars \(\) const](#)
- const T1 & [currentDelta \(\) const](#)
- const carl::FastMap< [FormulaT](#), std::vector< const [Bound< T1, T2 > \\*constraintToBound \(\) const](#)
- carl::FastMap< [FormulaT](#), std::vector< const [Bound< T1, T2 > \\*rConstraintToBound \(\)](#)
- size\_t [numberOfPivotingSteps \(\) const](#)
- void [resetNumberOfPivotingSteps \(\)](#)
- carl::FastPointerMap< [Variable< T1, T2 >](#), [LearnedBound >](#) & [rLearnedLowerBounds \(\)](#)
- carl::FastPointerMap< [Variable< T1, T2 >](#), [LearnedBound >](#) & [rLearnedUpperBounds \(\)](#)
- void [resetTheta \(\)](#)
- std::vector< typename carl::FastPointerMap< [Variable< T1, T2 >](#), [LearnedBound >](#)::iterator > & [rNewLearnedBounds \(\)](#)
- bool [entryIsPositive \(const \[TableauEntry< T1, T2 >\]\(#\) & \\_entry\) const](#)
- bool [entryIsNegative \(const \[TableauEntry< T1, T2 >\]\(#\) & \\_entry\) const](#)
- auto [defaultBoundPosition \(\) const](#)
- bool [isActive \(const \[Variable< T1, T2 >\]\(#\) & \\_var\) const](#)
- [EntryID newTableauEntry \(const T2 & \\_content\)](#)
- void [removeEntry \(\[EntryID\]\(#\) \\_entryID\)](#)
- [Variable< T1, T2 > \\* getVariable \(const \[Poly\]\(#\) & \\_lhs, T1 & \\_factor, T1 & \\_boundValue\)](#)
- [Variable< T1, T2 > \\* getObjectiveVariable \(const \[Poly\]\(#\) & \\_lhs\)](#)

- std::pair< const Bound< T1, T2 > \*, bool > newBound (const FormulaT &\_constraint)
- void removeBound (const FormulaT &\_constraint)
- void activateBound (const Bound< T1, T2 > \*\_bound, const FormulaT &\_formula)
- void deleteVariable (Variable< T1, T2 > \*\_variable, bool \_optimizationVar=false)
- Variable< T1, T2 > \* newNonbasicVariable (const typename Poly::PolyType \*\_poly, bool \_isInteger)
- Variable< T1, T2 > \* newBasicVariable (const typename Poly::PolyType \*\_poly, bool \_isInteger)
- void activateBasicVar (Variable< T1, T2 > \*\_var)
  - g
- void deactivateBasicVar (Variable< T1, T2 > \*\_var)
- void storeAssignment ()
- void resetAssignment ()
- RationalAssignment getRationalAssignment () const
- void adaptDelta (const Variable< T1, T2 > &\_variable, bool \_upperBound, T1 &\_minDelta) const
- void compressRows ()
- bool usedBlandsRule ()
  - Returns true, if the next pivoting element is selected by blands rule.*
- bool hasMultilineConflict ()
  - std::vector< const Bound< T1, T2 > \* > getMultilineConflict ()
  - std::pair< EntryID, bool > nextPivotingElement ()
  - Value< T1 > violationSum ()
    - Method to compute the sum of derivations to the closest bounds.*
- Value< T1 > dViolationSum (const Variable< T1, T2 > \*nVar, const Value< T1 > &update)
  - Computes the change in the violation sum when an the update value is added to the assignment of the nonbasic variable nVar.*
- Value< T1 > get\_dVioSum ()
  - Used as assertion-function to check the progress of the tableau.*
- std::map< Value< T1 >, Value< T1 > > compute\_dVio (const std::vector< std::pair< Value< T1 >, Variable< T1, T2 > \* > > &candidates, const Variable< T1, T2 > &nVar, bool positive)
  - Updated Method to computed dVio more efficiently.*
- std::pair< EntryID, bool > nextPivotingElementInfeasibilities ()
  - Method for next pivoting element according to "Simplex with Sum of Infeasibilities for SMT" by Tim King and Clark Barrett.*
- void computeLeavingCandidates (const std::size\_t i, std::vector< std::pair< Value< T1 >, Variable< T1, T2 > \* > > &leaving\_candidates)
  - Helper method to compute leavind candidates used in nextPivotingElementInfeasibilities.*
- void updateNonbasicVariable (EntryID \_pivotingElement)
  - Updates a nonbasic variable by the assignment stored in mpTheta and triggers updates of depending basic variables.*
- void updateNonbasicVariable (EntryID \_pivotingElement, Value< T1 > update)
  - Updates a nonbasic variable by the given assignment and triggers updates of depending basic variables.*
- std::pair< EntryID, bool > hasConflict ()
  - Checks if a conflicting pair exists.*
- void printEntries ()
  - std::vector< T2 > getInfeasibilityRow ()
    - Method to compute the infeasibility row to the current tableau.*
  - void setInfeasibilityRow ()
    - Method to set the initial infeasibility row to avoid recomputation in every round.*
- std::pair< EntryID, bool > optimizeIndependentNonbasics (const Variable< T1, T2 > &\_objective)
  - std::pair< EntryID, bool > nextPivotingElementForOptimizing (const Variable< T1, T2 > &\_objective)
  - std::pair< EntryID, bool > nextZeroPivotingElementForOptimizing (const Variable< T1, T2 > &\_objective)
    - const
  - EntryID isSuitable (const Variable< T1, T2 > &\_basicVar, bool \_forIncreasingAssignment) const
  - EntryID isSuitableConflictDetection (const Variable< T1, T2 > &\_basicVar, bool \_forIncreasingAssignment)
    - const

- Checks if the basic variable is pivotable.*
- bool `betterEntry (EntryID _isBetter, EntryID _than) const`
  - std::vector< const Bound< T1, T2 > \* > `getConflict (EntryID _rowEntry) const`
  - std::vector< std::vector< const Bound< T1, T2 > \* > > `getConflictsFrom (EntryID _rowEntry) const`
  - void `updateBasicAssignments (size_t _column, const Value< T1 > &_change)`
  - Variable< T1, T2 > \* `pivot (EntryID _pivotingElement, bool _optimizing=false)`
  - void `update (bool _downwards, EntryID _pivotingElement, std::vector< Iterator > &_pivotingRowLeftSide, std::vector< Iterator > &_pivotingRowRightSide, bool _optimizing=false)`
- Updates the tableau according to the new values in the pivoting row containing the given pivoting element.*
- void `addToEntry (const T2 &_toAdd, Iterator &_horilter, bool _horilterLeftFromVertlter, Iterator &_vertlter, bool _vertlterBelowHorilter)`
- Adds the given value to the entry being at the position  $(i,j)$ , where  $i$  is the vertical position of the given horizontal iterator and  $j$  is the horizontal position of the given vertical iterator.*
- void `rowRefinement (Variable< T1, T2 > *_basicVar)`

*Tries to refine the supremum and infimum of the given basic variable.*

    - size\_t `boundedVariables (const Variable< T1, T2 > &_var, size_t _stopCriterium=0) const`
    - size\_t `unboundedVariables (const Variable< T1, T2 > &_var, size_t _stopCriterium=0) const`
    - size\_t `checkCorrectness () const`
    - bool `rowCorrect (size_t _rowNumber) const`
    - bool `isConflicting () const`
    - const Poly::PolyType \* `gomoryCut (const T2 &_ass, Variable< T1, T2 > *_rowVar)`

*Creates a constraint referring to Gomory Cuts, if possible.*

      - size\_t `getNumberOfEntries (Variable< T1, T2 > *_rowVar)`
      - void `collect_premises (const Variable< T1, T2 > *_rowVar, FormulasT &premises) const`

*Collects the premises for branch and bound and stores them in premises.*

        - void `printHeap (std::ostream &_out=std::cout, int _maxEntryLength=30, const std::string _init="") const`
        - void `printEntry (EntryID _entry, std::ostream &_out=std::cout, int _maxEntryLength=20) const`
        - void `printVariables (bool _allBounds=true, std::ostream &_out=std::cout, const std::string _init="") const`
        - void `printLearnedBounds (const std::string _init="", std::ostream &_out=std::cout) const`
        - void `printLearnedBound (const Variable< T1, T2 > &_var, const LearnedBound &_learnedBound, const std::string _init="", std::ostream &_out=std::cout) const`
        - void `print (EntryID _pivotingElement=LAST_ENTRY_ID, std::ostream &_out=std::cout, const std::string _init="", bool _friendlyNames=true, bool _withZeroColumns=false) const`

### 0.15.536.1 Constructor & Destructor Documentation

**0.15.536.1.1 Tableau()** template<class Settings , typename T1 , typename T2 >  
`smtrat::lra::Tableau< Settings, T1, T2 >::Tableau (`  
`ModuleInput::iterator _defaultBoundPosition )`

#### Parameters

|                                    |
|------------------------------------|
| <code>_defaultBoundPosition</code> |
|------------------------------------|

**0.15.536.1.2 ~Tableau()** template<class Settings , typename T1 , typename T2 >  
`smtrat::lra::Tableau< Settings, T1, T2 >::~Tableau ( )`

### 0.15.536.2 Member Function Documentation

```
0.15.536.2.1 activateBasicVar() template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::activateBasicVar (
 Variable< T1, T2 > * _var)
g
```

**Parameters**

|      |  |
|------|--|
| _var |  |
|------|--|

```
0.15.536.2.2 activateBound() template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::activateBound (
 const Bound< T1, T2 > * _bound,
 const FormulaT & _formula)
```

**Parameters**

|          |  |
|----------|--|
| _bound   |  |
| _formula |  |

```
0.15.536.2.3 adaptDelta() template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::adaptDelta (
 const Variable< T1, T2 > & _variable,
 bool _upperBound,
 T1 & _minDelta) const
```

```
0.15.536.2.4 addToEntry() template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::addToEntry (
 const T2 & _toAdd,
 Iterator & _horIter,
 bool _horIterLeftFromVertIter,
 Iterator & _vertIter,
 bool _vertIterBelowHorIter)
```

Adds the given value to the entry being at the position (i,j), where i is the vertical position of the given horizontal iterator and j is the horizontal position of the given vertical iterator.

Note, that the entry might not exist, if its current value is 0. Then the horizontal iterator is located horizontally before or after the entry to change and the vertical iterator is located vertically before or after the entry to add.

**Parameters**

|                          |                                                                                                                                                                                                                                                                    |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| _toAdd                   | The value to add to the content of the entry specified by the given iterators and their relative position to each other.                                                                                                                                           |
| _horIter                 | The iterator moving horizontally and, hence, giving the vertical position of the entry to add the given value to.                                                                                                                                                  |
| _horIterLeftFromVertIter | true, if the horizontally moving iterator is left from or equal to the horizontal position of the iterator moving vertically, and, hence, left from or equal to the position of the entry to add the given value to; false, it is right or equal to this position. |
| _vertIter                | The iterator moving vertically and, hence, giving the horizontal position of the entry to add the given value to.                                                                                                                                                  |

**Parameters**

|                                    |                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_vertIterBelowHorIter</code> | true, if the vertically moving iterator is below or exactly at the vertical position of the iterator moving horizontally, and, hence, below or exactly at the position of the entry to add the given value to; false, it is above or equal to this position. @sideeffect If the entry existed (!=0) and is removed because of becoming 0, the iterators are set according to the given relative positioning. |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```
0.15.536.2.5 betterEntry() template<class Settings , typename T1 , typename T2 >
bool smtrat::lra::Tableau< Settings, T1, T2 >::betterEntry (
 EntryID _isBetter,
 EntryID _than) const
```

**Parameters**

|                        |  |
|------------------------|--|
| <code>_isBetter</code> |  |
| <code>_than</code>     |  |

**Returns**

```
0.15.536.2.6 boundedVariables() template<class Settings , typename T1 , typename T2 >
size_t smtrat::lra::Tableau< Settings, T1, T2 >::boundedVariables (
 const Variable< T1, T2 > & _var,
 size_t _stopCriterium = 0) const
```

**Parameters**

|                             |  |
|-----------------------------|--|
| <code>_var</code>           |  |
| <code>_stopCriterium</code> |  |

**Returns**

```
0.15.536.2.7 checkCorrectness() template<class Settings , typename T1 , typename T2 >
size_t smtrat::lra::Tableau< Settings, T1, T2 >::checkCorrectness () const
```

**Returns**

```
0.15.536.2.8 collect_premises() template<class Settings , typename T1 , typename T2 >
void smtrat::lra::Tableau< Settings, T1, T2 >::collect_premises (
 const Variable< T1, T2 > * _rowVar,
 FormulasT & premises) const
```

Collects the premises for branch and bound and stores them in premises.

## Parameters

|                       |  |
|-----------------------|--|
| <code>_rowVar</code>  |  |
| <code>premises</code> |  |

**0.15.536.2.9 `columns()`** template<class Settings , typename T1 , typename T2 >  
 const std::vector<Variable<T1, T2>\*>& smtrat::lra::Tableau< Settings, T1, T2 >::columns ( )  
 const [inline]

## Returns

**0.15.536.2.10 `compressRows()`** template<class Settings , typename T1 , typename T2 >  
 void smtrat::lra::Tableau< Settings, T1, T2 >::compressRows ( )

**0.15.536.2.11 `compute_dVio()`** template<class Settings , typename T1 , typename T2 >  
 std::map<Value<T1>, Value<T1> > smtrat::lra::Tableau< Settings, T1, T2 >::compute\_dVio (   
     const std::vector< std::pair< Value< T1 >, Variable< T1, T2 > \* > > & candidates,  
     const Variable< T1, T2 > & nVar,  
     bool positive )

Updated Method to computed dVio more efficienty.

For this the fact is used, that VioSum is piecewise linear (the proof can be found in the Sol paper).

## Parameters

|                         |                                                                                |
|-------------------------|--------------------------------------------------------------------------------|
| <code>candidates</code> | Update candidates, IMPORTANT: Must be sorted with an stabloe sorting algorithm |
| <code>nVar</code>       | The updated nonbasic variable                                                  |
| <code>positive</code>   | Whether the positve or negative update values are checked                      |

**0.15.536.2.12 `computeLeavingCandidates()`** template<class Settings , typename T1 , typename T2 >

```
void smtrat::lra::Tableau< Settings, T1, T2 >::computeLeavingCandidates (

 const std::size_t i,

 std::vector< std::pair< Value< T1 >, Variable< T1, T2 > * > > & leaving_<-

 candidates)
```

Helper method to compute leavind candidates used in nextPivotingElementInfeasibilities.

Computes for every non-basic variable the breaking points. Updates from non-basic vars of 0 are omitted to prevent loops.

## Parameters

|                                 |                                                 |
|---------------------------------|-------------------------------------------------|
| <code>i</code>                  | : Index if entering (nonbasic ) variable.       |
| <code>leaving_candidates</code> | : Used as storage object for update candidates. |

**0.15.536.2.13 `constraintToBounds()`** template<class Settings , typename T1 , typename T2 >  
 const carl::FastMap<FormulaT, std::vector<const Bound<T1, T2>\*>>& smtrat::lra::Tableau< Settings, T1, T2 >::constraintToBounds ( ) const [inline]

Returns

**0.15.536.2.14 currentDelta()** template<class Settings , typename T1 , typename T2 >  
const T1& smtrat::lra::Tableau< Settings, T1, T2 >::currentDelta ( ) const [inline]

**0.15.536.2.15 deactivateBasicVar()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::deactivateBasicVar (  
    Variable< T1, T2 > \* \_var )

Parameters

|      |  |
|------|--|
| _var |  |
|------|--|

**0.15.536.2.16 defaultBoundPosition()** template<class Settings , typename T1 , typename T2 >  
auto smtrat::lra::Tableau< Settings, T1, T2 >::defaultBoundPosition ( ) const [inline]

Returns

**0.15.536.2.17 deleteVariable()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::deleteVariable (  
    Variable< T1, T2 > \* \_variable,  
    bool \_optimizationVar = false )

Parameters

|           |  |
|-----------|--|
| _variable |  |
|-----------|--|

**0.15.536.2.18 dViolationSum()** template<class Settings , typename T1 , typename T2 >  
Value<T1> smtrat::lra::Tableau< Settings, T1, T2 >::dViolationSum (  
    const Variable< T1, T2 > \* nVar,  
    const Value< T1 > & update )

Computes the change in the violation sum when an the update value is added to the assignment of the nonbassi variable nVar.

Parameters

|        |                        |
|--------|------------------------|
| nVar   | variable to be updated |
| update | Update value           |

**0.15.536.2.19 entryIsNegative()** template<class Settings , typename T1 , typename T2 >  
bool smtrat::lra::Tableau< Settings, T1, T2 >::entryIsNegative (  
    const TableauEntry< T1, T2 > & \_entry ) const [inline]

```
0.15.536.2.20 entryIsPositive() template<class Settings , typename T1 , typename T2 >
bool smtrat::lra::Tableau< Settings, T1, T2 >::entryIsPositive (
 const TableauEntry< T1, T2 > & _entry) const [inline]
```

**0.15.536.2.21 get\_dVioSum()** template<class Settings , typename T1 , typename T2 >

```
Value<T1> smtrat::lra::Tableau< Settings, T1, T2 >::get_dVioSum () [inline]
```

Used as assertion-function to check the progress of the tableau.

```
0.15.536.2.22 getConflict() template<class Settings , typename T1 , typename T2 >
std::vector< const Bound<T1, T2>* > smtrat::lra::Tableau< Settings, T1, T2 >::getConflict (
 EntryID _rowEntry) const
```

Parameters

|           |
|-----------|
| _rowEntry |
|-----------|

Returns

```
0.15.536.2.23 getConflictsFrom() template<class Settings , typename T1 , typename T2 >
std::vector< std::vector< const Bound<T1, T2>* > > smtrat::lra::Tableau< Settings, T1, T2 >::getConflictsFrom (
 EntryID _rowEntry) const
```

Parameters

|           |
|-----------|
| _rowEntry |
|-----------|

Returns

```
0.15.536.2.24 getInfeasibilityRow() template<class Settings , typename T1 , typename T2 >
std::vector<T2> smtrat::lra::Tableau< Settings, T1, T2 >::getInfeasibilityRow ()
```

Method to compute the infeasibility row to the current tableau.

```
0.15.536.2.25 getMultilineConflict() template<class Settings , typename T1 , typename T2 >
std::vector< const Bound<T1, T2>* > smtrat::lra::Tableau< Settings, T1, T2 >::getMultilineConflict ()
```

Returns

In SUM-Simplex a conflict is possibly created with multiple rows. This function returns the single conflict if possible and otherwise construct the multiline conflict.

```
0.15.536.2.26 getNumberOfEntries() template<class Settings , typename T1 , typename T2 >
size_t smtrat::lra::Tableau< Settings, T1, T2 >::getNumberOfEntries (
 Variable< T1, T2 > * _rowVar)
```

**Parameters**

|                      |  |
|----------------------|--|
| <code>_rowVar</code> |  |
|----------------------|--|

**Returns**

number of entries in the row belonging to `_rowVar`

**0.15.536.2.27 `getObjectiveVariable()`** template<class `Settings` , typename `T1` , typename `T2` >  
`Variable<T1,T2>* smtrat::lra::Tableau< Settings, T1, T2 >::getObjectiveVariable (`  
`const Poly & _lhs )`

**0.15.536.2.28 `getRationalAssignment()`** template<class `Settings` , typename `T1` , typename `T2` >  
`RationalAssignment smtrat::lra::Tableau< Settings, T1, T2 >::getRationalAssignment ( ) const`

**Returns**

**0.15.536.2.29 `getVariable()`** template<class `Settings` , typename `T1` , typename `T2` >  
`Variable<T1,T2>* smtrat::lra::Tableau< Settings, T1, T2 >::getVariable (`  
`const Poly & _lhs,`  
`T1 & _factor,`  
`T1 & _boundValue )`

**0.15.536.2.30 `gomoryCut()`** template<class `Settings` , typename `T1` , typename `T2` >  
const `Poly::PolyType* smtrat::lra::Tableau< Settings, T1, T2 >::gomoryCut (`  
`const T2 & _ass,`  
`Variable< T1, T2 > * _rowVar )`

Creates a constraint referring to Gomory Cuts, if possible.

**Parameters**

|                      |  |
|----------------------|--|
| <code>_ass</code>    |  |
| <code>_rowVar</code> |  |

**Returns**

NULL, if the cut can't be constructed; otherwise the valid constraint is returned.

**0.15.536.2.31 `hasConflict()`** template<class `Settings` , typename `T1` , typename `T2` >  
`std::pair<EntryID,bool> smtrat::lra::Tableau< Settings, T1, T2 >::hasConflict ( )`

Checks if a conflicting pair exists.

In a conflict is found, the cell and false is returned. If not, LAST\_ENTRY\_ID and true is returned.

**0.15.536.2.32 `hasMultilineConflict()`** template<class `Settings` , typename `T1` , typename `T2` >  
`bool smtrat::lra::Tableau< Settings, T1, T2 >::hasMultilineConflict ( )`

**Returns**

In SUM-Simplex a conflict is possibly created with multiple rows. This function returns true iff the multi-conflict settings occurred.

**0.15.536.2.33 isActive()** template<class Settings , typename T1 , typename T2 >  
 bool smtrat::lra::Tableau< Settings, T1, T2 >::isActive ( const Variable< T1, T2 > & \_var ) const [inline]

**0.15.536.2.34 isConflicting()** template<class Settings , typename T1 , typename T2 >  
 bool smtrat::lra::Tableau< Settings, T1, T2 >::isConflicting ( ) const

**0.15.536.2.35 isSuitable()** template<class Settings , typename T1 , typename T2 >  
 EntryID smtrat::lra::Tableau< Settings, T1, T2 >::isSuitable ( const Variable< T1, T2 > & \_basicVar, bool \_forIncreasingAssignment ) const

**Parameters**

|                                 |  |
|---------------------------------|--|
| <i>_basicVar</i>                |  |
| <i>_forIncreasingAssignment</i> |  |

**Returns**

**0.15.536.2.36 isSuitableConflictDetection()** template<class Settings , typename T1 , typename T2 >  
 EntryID smtrat::lra::Tableau< Settings, T1, T2 >::isSuitableConflictDetection ( const Variable< T1, T2 > & \_basicVar, bool \_forIncreasingAssignment ) const

Checks if the basic variable is pivotable.

Returns LAST\_ENTRY\_ID as EntryID if a conflict was found. In difference to isSuitable returns this function an EntryID as soon as the EntryID bestEntry is unequal to LAST\_ENTRY\_ID. The assertions are still checked.

**Parameters**

|                                 |  |
|---------------------------------|--|
| <i>_basicVar</i>                |  |
| <i>_forIncreasingAssignment</i> |  |

**Returns**

**0.15.536.2.37 newBasicVariable()** template<class Settings , typename T1 , typename T2 >  
 Variable<T1, T2>\* smtrat::lra::Tableau< Settings, T1, T2 >::newBasicVariable ( const typename Poly::PolyType \* \_poly, bool \_isInteger )

**Parameters**

|                      |  |
|----------------------|--|
| <i>_poly</i>         |  |
| <i>_originalVars</i> |  |
| <i>_isInteger</i>    |  |

**Returns**

**0.15.536.2.38 newBound()** template<class Settings , typename T1 , typename T2 >  
std::pair<const Bound<T1,T2>\*, bool> smtrat::lra::Tableau< Settings, T1, T2 >::newBound (  
    const FormulaT & *\_constraint* )

**Parameters**

|                    |  |
|--------------------|--|
| <i>_constraint</i> |  |
|--------------------|--|

**Returns**

**0.15.536.2.39 newNonbasicVariable()** template<class Settings , typename T1 , typename T2 >  
Variable<T1, T2>\* smtrat::lra::Tableau< Settings, T1, T2 >::newNonbasicVariable (  
    const typename Poly::PolyType \* *\_poly*,  
    bool *\_isInteger* )

**Parameters**

|                   |  |
|-------------------|--|
| <i>_poly</i>      |  |
| <i>_isInteger</i> |  |

**Returns**

**0.15.536.2.40 newTableauEntry()** template<class Settings , typename T1 , typename T2 >  
EntryID smtrat::lra::Tableau< Settings, T1, T2 >::newTableauEntry (  
    const T2 & *\_content* )

**Parameters**

|                 |  |
|-----------------|--|
| <i>_content</i> |  |
|-----------------|--|

**Returns**

**0.15.536.2.41 nextPivotingElement()** template<class Settings , typename T1 , typename T2 >  
std::pair<EntryID, bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextPivotingElement ()

Returns

```
0.15.536.2.42 nextPivotingElementForOptimizing() template<class Settings , typename T1 , typename T2 >
std::pair<EntryID,bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextPivotingElementForOptimizing (
 const Variable< T1, T2 > & _objective)
```

Returns

```
0.15.536.2.43 nextPivotingElementInfeasibilities() template<class Settings , typename T1 , typename T2 >
std::pair<EntryID, bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextPivotingElementInfeasibilities ()
```

Method for next pivoting element according to "Simplex with Sum of Infeasibilities for SMT" by Tim King and Clark Barrett.

Therefore, one counts the violation of every bound and searches for elements who minimize the violation sum of the next iteration.

Returns

```
0.15.536.2.44 nextZeroPivotingElementForOptimizing() template<class Settings , typename T1 , typename T2 >
std::pair<EntryID,bool> smtrat::lra::Tableau< Settings, T1, T2 >::nextZeroPivotingElementForOptimizing (
 const Variable< T1, T2 > & _objective) const
```

```
0.15.536.2.45 numberOfPivotingSteps() template<class Settings , typename T1 , typename T2 >
size_t smtrat::lra::Tableau< Settings, T1, T2 >::numberOfPivotingSteps () const [inline]
```

Returns

```
0.15.536.2.46 optimizeIndependentNonbasics() template<class Settings , typename T1 , typename T2 >
std::pair<EntryID,bool> smtrat::lra::Tableau< Settings, T1, T2 >::optimizeIndependentNonbasics (
 const Variable< T1, T2 > & _objective)
```

Parameters

|            |
|------------|
| _objective |
|------------|

Returns

**0.15.536.2.47 originalVars()** template<class Settings , typename T1 , typename T2 >  
const carl::FastMap< carl::Variable, Variable<T1,T2>\*>& smtrat::lra::Tableau< Settings, T1,  
T2 >::originalVars ( ) const [inline]

Returns

**0.15.536.2.48 pivot()** template<class Settings , typename T1 , typename T2 >  
Variable<T1, T2>\* smtrat::lra::Tableau< Settings, T1, T2 >::pivot (  
    EntryID \_pivotingElement,  
    bool \_optimizing = false )

Parameters

|                          |  |
|--------------------------|--|
| <i>_pivotingElement</i>  |  |
| <i>updateAssignments</i> |  |

Returns

**0.15.536.2.49 print()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::print (  
    EntryID \_pivotingElement = LAST\_ENTRY\_ID,  
    std::ostream & \_out = std::cout,  
    const std::string \_init = "",  
    bool \_friendlyNames = true,  
    bool \_withZeroColumns = false ) const

Parameters

|                         |  |
|-------------------------|--|
| <i>_pivotingElement</i> |  |
| <i>_out</i>             |  |
| <i>_init</i>            |  |
| <i>_friendlyNames</i>   |  |
| <i>_withZeroColumns</i> |  |

**0.15.536.2.50 printEntries()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::printEntries ( ) [inline]

**0.15.536.2.51 printEntry()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::printEntry (  
    EntryID \_entry,

```
std::ostream & _out = std::cout,
int _maxEntryLength = 20) const
```

**Parameters**

|                 |  |
|-----------------|--|
| _entry          |  |
| _out            |  |
| _maxEntryLength |  |

**0.15.536.2.52 printHeap()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::printHeap (

```
std::ostream & _out = std::cout,
int _maxEntryLength = 30,
const std::string _init = "") const
```

**Parameters**

|                 |  |
|-----------------|--|
| _out            |  |
| _maxEntryLength |  |
| _init           |  |

**0.15.536.2.53 printLearnedBound()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::printLearnedBound (

```
const Variable< T1, T2 > & _var,
const LearnedBound & _learnedBound,
const std::string _init = "",
std::ostream & _out = std::cout) const
```

**Parameters**

|               |  |
|---------------|--|
| _var          |  |
| _learnedBound |  |
| _init         |  |
| _out          |  |

**0.15.536.2.54 printLearnedBounds()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::printLearnedBounds (

```
const std::string _init = "",
std::ostream & _out = std::cout) const
```

**Parameters**

|       |  |
|-------|--|
| _init |  |
| _out  |  |

**0.15.536.2.55 printVariables()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::printVariables (

```
bool _allBounds = true,
```

```
 std::ostream & _out = std::cout,
 const std::string _init = "") const
```

**Parameters**

|                         |  |
|-------------------------|--|
| <code>_allBounds</code> |  |
| <code>_out</code>       |  |
| <code>_init</code>      |  |

**0.15.536.2.56 `rConstraintToBound()`** template<class Settings , typename T1 , typename T2 >  
carl::FastMap<FormulaT, std::vector<const Bound<T1, T2>\*>>& smtrat::lra::Tableau< Settings,  
T1, T2 >::rConstraintToBound ( ) [inline]

**Returns**

**0.15.536.2.57 `removeBound()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::removeBound (   
 const FormulaT & \_constraint )

**0.15.536.2.58 `removeEntry()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::removeEntry (   
 EntryID \_entryID )

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_entryID</code> |  |
|-----------------------|--|

**0.15.536.2.59 `resetAssignment()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::resetAssignment ( )

**0.15.536.2.60 `resetNumberOfPivotingSteps()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::resetNumberOfPivotingSteps ( ) [inline]

**0.15.536.2.61 `resetTheta()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::resetTheta ( ) [inline]

**0.15.536.2.62 `rLearnedLowerBounds()`** template<class Settings , typename T1 , typename T2 >  
carl::FastPointerMap<Variable<T1,T2>, LearnedBound>& smtrat::lra::Tableau< Settings, T1, T2  
>::rLearnedLowerBounds ( ) [inline]

**Returns**

**0.15.536.2.63 `rLearnedUpperBounds()`** template<class Settings , typename T1 , typename T2 >  
carl::FastPointerMap<Variable<T1,T2>, LearnedBound>& smtrat::lra::Tableau< Settings, T1, T2  
>::rLearnedUpperBounds ( ) [inline]

Returns

**0.15.536.2.64 `rNewLearnedBounds()`** template<class Settings , typename T1 , typename T2 >  
std::vector<typename carl::FastPointerMap<Variable<T1,T2>, LearnedBound>::iterator>& smtrat::lra::Tableau<  
Settings, T1, T2 >::rNewLearnedBounds ( ) [inline]

Returns

**0.15.536.2.65 `rowCorrect()`** template<class Settings , typename T1 , typename T2 >  
bool smtrat::lra::Tableau< Settings, T1, T2 >::rowCorrect (   
size\_t \_rowNumber ) const

Parameters

|                         |
|-------------------------|
| <code>_rowNumber</code> |
|-------------------------|

Returns

**0.15.536.2.66 `rowRefinement()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::rowRefinement (   
Variable< T1, T2 > \* \_basicVar )

Tries to refine the supremum and infimum of the given basic variable.

Parameters

|                        |                                                                  |
|------------------------|------------------------------------------------------------------|
| <code>_basicVar</code> | The basic variable for which to refine the supremum and infimum. |
|------------------------|------------------------------------------------------------------|

**0.15.536.2.67 `rows()`** template<class Settings , typename T1 , typename T2 >  
const std::vector<Variable<T1, T2>\*>& smtrat::lra::Tableau< Settings, T1, T2 >::rows ( )  
const [inline]

Returns

**0.15.536.2.68 `setBlandsRuleStart()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::setBlandsRuleStart (   
size\_t \_start ) [inline]

**0.15.536.2.69 `setInfeasibilityRow()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::setInfeasibilityRow ( )  
Method to set the initial infeasibility row to avoid recomputation in every round.

**0.15.536.2.70 `setSize()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::setSize (   
    size\_t \_expectedHeight,  
    size\_t \_expectedWidth,  
    size\_t \_expectedNumberOfBounds ) [inline]

**Parameters**

|                                |  |
|--------------------------------|--|
| <i>_expectedHeight</i>         |  |
| <i>_expectedWidth</i>          |  |
| <i>_expectedNumberOfBounds</i> |  |

**0.15.536.2.71 `size()`** template<class Settings , typename T1 , typename T2 >  
size\_t smtrat::lra::Tableau< Settings, T1, T2 >::size ( ) const [inline]

**Returns**

**0.15.536.2.72 `slackVars()`** template<class Settings , typename T1 , typename T2 >  
const carl::FastPointerMap<typename Poly::PolyType, Variable<T1,T2>\*>& smtrat::lra::Tableau< Settings, T1, T2 >::slackVars ( ) const [inline]

**Returns**

**0.15.536.2.73 `storeAssignment()`** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::storeAssignment ( )

**0.15.536.2.74 `unboundedVariables()`** template<class Settings , typename T1 , typename T2 >  
size\_t smtrat::lra::Tableau< Settings, T1, T2 >::unboundedVariables (   
    const Variable< T1, T2 > & \_var,  
    size\_t \_stopCriterium = 0 ) const

**Parameters**

|                       |  |
|-----------------------|--|
| <i>_var</i>           |  |
| <i>_stopCriterium</i> |  |

**Returns**

**0.15.536.2.75 `update()`** template<class Settings , typename T1 , typename T2 >

```
void smtrat::lra::Tableau< Settings, T1, T2 >::update (
 bool _downwards,
 EntryID _pivotingElement,
 std::vector< Iterator > & _pivotingRowLeftSide,
 std::vector< Iterator > & _pivotingRowRightSide,
 bool _optimizing = false)
```

Updates the tableau according to the new values in the pivoting row containing the given pivoting element.  
The updating is applied from the pivoting row downwards, if the given flag `_downwards` is true, and upwards, otherwise.

#### Parameters

|                                    |                                                                                                                                                                                                                                   |
|------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_downwards</code>            | The flag indicating whether to update the tableau downwards or upwards starting from the pivoting row.                                                                                                                            |
| <code>_pivotingElement</code>      | The id of the current pivoting element.                                                                                                                                                                                           |
| <code>_pivotingRowLeftSide</code>  | For every element in the pivoting row, which is positioned left of the pivoting element, this vector contains an iterator. The closer the element is to the pivoting element, the smaller is the iterator's index in the vector.  |
| <code>_pivotingRowRightSide</code> | For every element in the pivoting row, which is positioned right of the pivoting element, this vector contains an iterator. The closer the element is to the pivoting element, the smaller is the iterator's index in the vector. |
| <code>_updateAssignments</code>    | If true, the assignments of all variables will be updated after pivoting.                                                                                                                                                         |

**0.15.536.2.76 updateBasicAssignments()** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::updateBasicAssignments (  
 size\_t \_column,  
 const Value< T1 > & \_change )

#### Parameters

|                      |  |
|----------------------|--|
| <code>_column</code> |  |
| <code>_change</code> |  |

**0.15.536.2.77 updateNonbasicVariable() [1/2]** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::updateNonbasicVariable (  
 EntryID \_pivotingElement )

Updates a nonbasic variable by the assignment stored in mpTheta and triggers updates of depending basic variables.

#### Parameters

|                               |                                             |
|-------------------------------|---------------------------------------------|
| <code>_pivotingElement</code> | Element in the row of the nonbasic variable |
|-------------------------------|---------------------------------------------|

**0.15.536.2.78 updateNonbasicVariable() [2/2]** template<class Settings , typename T1 , typename T2 >  
void smtrat::lra::Tableau< Settings, T1, T2 >::updateNonbasicVariable (  
 EntryID \_pivotingElement,  
 Value< T1 > update )

Updates a nonbasic variable by the given assignment and triggers updates of depending basic variables.

**Parameters**

|                         |                                               |
|-------------------------|-----------------------------------------------|
| <i>_pivotingElement</i> | Element in the row of the nonbasic variable   |
| <i>update</i>           | Update <a href="#">Value</a> of non-basic var |

**0.15.536.2.79 [usedBlndsRule\(\)](#)** template<class Settings , typename T1 , typename T2 >  
bool smtrat::lra::Tableau< Settings, T1, T2 >::usedBlndsRule ( )  
Returns true, if the next pivoting element is selected by blnds rule.

**0.15.536.2.80 [violationSum\(\)](#)** template<class Settings , typename T1 , typename T2 >  
[Value](#)<T1> smtrat::lra::Tableau< Settings, T1, T2 >::violationSum ( )  
Method to compute the sum of derivations to the closest bounds.

## 0.15.537 [smtrat::lra::TableauEntry< T1, T2 >](#) Class Template Reference

```
#include <Tableau.h>
```

### Public Member Functions

- [TableauEntry \(\)](#)
- [TableauEntry \(EntryID \\_up, EntryID \\_down, EntryID \\_left, EntryID \\_right, Variable< T1, T2 > \\*\\_rowVar, Variable< T1, T2 > \\*\\_columnVar, const T2 & \\_content\)](#)
- [TableauEntry \(const TableauEntry &\\_entry\)](#)
- [~TableauEntry \(\)](#)
- void [setVNext \(bool downwards, const EntryID \\_entryId\)](#)
- void [setHNext \(bool leftwards, const EntryID \\_entryId\)](#)
- [EntryID vNext \(bool downwards\)](#)
- [EntryID hNext \(bool leftwards\)](#)
- [Variable< T1, T2 > \\* rowVar \(\) const](#)
- void [setRowVar \(Variable< T1, T2 > \\*\\_rowVar\)](#)
- [Variable< T1, T2 > \\* columnVar \(\) const](#)
- void [setColumnVar \(Variable< T1, T2 > \\*\\_columnVar\)](#)
- const T2 & [content \(\) const](#)
- T2 & [rContent \(\)](#)

### 0.15.537.1 Constructor & Destructor Documentation

**0.15.537.1.1 [TableauEntry\(\)](#) [1/3]** template<typename T1 , typename T2 >  
smtrat::lra::TableauEntry< T1, T2 >::TableauEntry ( ) [inline]

**0.15.537.1.2 [TableauEntry\(\)](#) [2/3]** template<typename T1 , typename T2 >  
smtrat::lra::TableauEntry< T1, T2 >::TableauEntry (   
    EntryID \_up,  
    EntryID \_down,  
    EntryID \_left,  
    EntryID \_right,  
    Variable< T1, T2 > \* \_rowVar,  
    Variable< T1, T2 > \* \_columnVar,  
    const T2 & \_content ) [inline]

**Parameters**

|                   |  |
|-------------------|--|
| <i>_up</i>        |  |
| <i>_down</i>      |  |
| <i>_left</i>      |  |
| <i>_right</i>     |  |
| <i>_rowVar</i>    |  |
| <i>_columnVar</i> |  |
| <i>_content</i>   |  |

**0.15.537.1.3 TableauEntry()** [3/3] template<typename T1 , typename T2 >  
`smtrat::lra::TableauEntry< T1, T2 >::TableauEntry (`  
`const TableauEntry< T1, T2 > & _entry ) [inline]`

**Parameters**

|               |  |
|---------------|--|
| <i>_entry</i> |  |
|---------------|--|

**0.15.537.1.4 ~TableauEntry()** template<typename T1 , typename T2 >  
`smtrat::lra::TableauEntry< T1, T2 >::~TableauEntry ( ) [inline]`

**0.15.537.2 Member Function Documentation**

**0.15.537.2.1 columnVar()** template<typename T1 , typename T2 >  
`Variable<T1, T2>* smtrat::lra::TableauEntry< T1, T2 >::columnVar ( ) const [inline]`

**Returns**

**0.15.537.2.2 content()** template<typename T1 , typename T2 >  
`const T2& smtrat::lra::TableauEntry< T1, T2 >::content ( ) const [inline]`

**Returns**

**0.15.537.2.3 hNext()** template<typename T1 , typename T2 >  
`EntryID smtrat::lra::TableauEntry< T1, T2 >::hNext (`  
`bool leftwards ) [inline]`

**Parameters**

|                  |  |
|------------------|--|
| <i>leftwards</i> |  |
|------------------|--|

**Returns**

**0.15.537.2.4 rContent()** template<typename T1 , typename T2 >  
T2& smtrat::lra::TableauEntry< T1, T2 >::rContent ( ) [inline]

**Returns**

**0.15.537.2.5 rowVar()** template<typename T1 , typename T2 >  
Variable<T1, T2>\* smtrat::lra::TableauEntry< T1, T2 >::rowVar ( ) const [inline]

**Returns**

**0.15.537.2.6 setColumnVar()** template<typename T1 , typename T2 >  
void smtrat::lra::TableauEntry< T1, T2 >::setColumnVar (  
    Variable< T1, T2 > \* \_columnVar ) [inline]

**Parameters**

|                         |  |
|-------------------------|--|
| <code>_columnVar</code> |  |
|-------------------------|--|

**0.15.537.2.7 setHNext()** template<typename T1 , typename T2 >  
void smtrat::lra::TableauEntry< T1, T2 >::setHNext (  
    bool *leftwards*,  
    const EntryID *entryId* ) [inline]

**Parameters**

|                        |  |
|------------------------|--|
| <code>leftwards</code> |  |
| <code>_entryId</code>  |  |

**0.15.537.2.8 setRowVar()** template<typename T1 , typename T2 >  
void smtrat::lra::TableauEntry< T1, T2 >::setRowVar (  
    Variable< T1, T2 > \* *rowVar* ) [inline]

**Parameters**

|                      |  |
|----------------------|--|
| <code>_rowVar</code> |  |
|----------------------|--|

**0.15.537.2.9 setVNext()** template<typename T1 , typename T2 >  
void smtrat::lra::TableauEntry< T1, T2 >::setVNext (  
    bool *downwards*,

```
const EntryID _entryId) [inline]
```

**Parameters**

|                  |  |
|------------------|--|
| <i>downwards</i> |  |
| <i>_entryId</i>  |  |

**0.15.537.2.10 vNext()** template<typename T1 , typename T2 >  
**EntryID smtrat::lra::TableauEntry< T1, T2 >::vNext (**  
**bool *downwards* ) [inline]**

**Parameters**

|                  |  |
|------------------|--|
| <i>downwards</i> |  |
|------------------|--|

**Returns****0.15.538 smtrat::cadcells::datastructures::TaggedIndexedRoot Struct Reference**

```
#include <delineation.h>
```

**Data Fields**

- [IndexedRoot root](#)
- bool [is\\_inclusive](#) = false
- bool [is\\_optional](#) = false
- std::optional<[PolyRef](#)> [origin](#) = std::nullopt

**0.15.538.1 Field Documentation**

**0.15.538.1.1 is\_inclusive** bool smtrat::cadcells::datastructures::TaggedIndexedRoot::is\_inclusive  
= false

**0.15.538.1.2 is\_optional** bool smtrat::cadcells::datastructures::TaggedIndexedRoot::is\_optional  
= false

**0.15.538.1.3 origin** std::optional<[PolyRef](#)> smtrat::cadcells::datastructures::TaggedIndexedRoot::origin  
= std::nullopt

**0.15.538.1.4 root** [IndexedRoot](#) smtrat::cadcells::datastructures::TaggedIndexedRoot::root

**0.15.539 smtrat::mcsat::onecellcad::TagPoly Struct Reference**

Tagged Polynomials.

```
#include <utils.h>
```

## Data Fields

- `InvarianceType tag`
- `Poly poly`
- `std::size_t level`
- `std::size_t deg = 0`

### 0.15.539.1 Detailed Description

Tagged Polynomials.

### 0.15.539.2 Field Documentation

**0.15.539.2.1 deg** `std::size_t smtrat::mcsat::onecellcad::TagPoly::deg = 0`

**0.15.539.2.2 level** `std::size_t smtrat::mcsat::onecellcad::TagPoly::level`

**0.15.539.2.3 poly** `Poly smtrat::mcsat::onecellcad::TagPoly::poly`

**0.15.539.2.4 tag** `InvarianceType smtrat::mcsat::onecellcad::TagPoly::tag`

## 0.15.540 smtrat::Task Class Reference

```
#include <ThreadPool.h>
```

### Public Member Functions

- `template<typename T> Task(T &&task, const Module *module, std::size_t _index)`
- `void run()`
- `const Module * getModule() const`
- `std::size_t conditionalIndex() const`
- `std::future<Answer> getFuture()`
- `bool operator<(const Task &rhs) const`

### 0.15.540.1 Constructor & Destructor Documentation

**0.15.540.1.1 Task()** `template<typename T> smtrat::Task::Task( T && task, const Module * module, std::size_t _index ) [inline]`

### 0.15.540.2 Member Function Documentation

**0.15.540.2.1 conditionalIndex()** `std::size_t smtrat::Task::conditionalIndex() const [inline]`

**0.15.540.2.2 `getFuture()`** std::future<[Answer](#)> smtrat::Task::getFuture ( ) [inline]

**0.15.540.2.3 `getModule()`** const [Module](#)\* smtrat::Task::getModule ( ) const [inline]

**0.15.540.2.4 `operator<()`** bool smtrat::Task::operator< ( const [Task](#) & rhs ) const

**0.15.540.2.5 `run()`** void smtrat::Task::run ( ) [inline]

## 0.15.541 [delta::TempFilenameGenerator Class Reference](#)

This class generates and reuses temporary filenames with a common prefix.

```
#include <utils.h>
```

### Public Member Functions

- [TempFilenameGenerator](#) (const std::string &prefix)  
*Constructor.*
- [~TempFilenameGenerator](#) ()  
*Destructor.*
- std::string [get](#) ()  
*Retrieve a filename for a temporary file that is not in use.*
- void [put](#) (const std::string &temp)  
*Returns a filename to the pool of available filenames.*

### 0.15.541.1 Detailed Description

This class generates and reuses temporary filenames with a common prefix.

### 0.15.541.2 Constructor & Destructor Documentation

**0.15.541.2.1 `TempFilenameGenerator()`** delta::TempFilenameGenerator::TempFilenameGenerator ( const std::string & prefix ) [inline]

Constructor.

#### Parameters

|               |                           |
|---------------|---------------------------|
| <i>prefix</i> | Prefix for all filenames. |
|---------------|---------------------------|

**0.15.541.2.2 `~TempFilenameGenerator()`** delta::TempFilenameGenerator::~TempFilenameGenerator ( ) [inline]

Destructor.

### 0.15.541.3 Member Function Documentation

**0.15.541.3.1 `get()`** std::string delta::TempFilenameGenerator::get ( ) [inline]

Retrieve a filename for a temporary file that is not in use.

**Returns**

Temporary filename.

**0.15.541.3.2 `put()`** `void delta::TempFilenameGenerator::put (`  
`const std::string & temp ) [inline]`

Returns a filename to the pool of available filenames.

**Parameters**

|                   |                     |
|-------------------|---------------------|
| <code>temp</code> | Temporary filename. |
|-------------------|---------------------|

## 0.15.542 smtrat::parser::TermParser Struct Reference

#include <Term.h>

**Public Types**

- `typedef conversion::VariantVariantConverter< types::TermType > Converter`

**Public Member Functions**

- `TermParser (Theories *theories)`

**Data Fields**

- `Theories * theories`
- `SymbolParser symbol`
- `SpecConstantParser speconstant`
- `QualifiedIdentifierParser qualifiedidentifier`
- `SortedVariableParser sortedvariable`
- `AttributeParser attribute`
- `Converter converter`
- `qi::rule< Iterator, Skipper > binding`
- `qi::rule< Iterator, types::TermType(), Skipper > termop`
- `qi::rule< Iterator, types::TermType(), Skipper > main`

### 0.15.542.1 Member Typedef Documentation

**0.15.542.1.1 `Converter`** `typedef conversion::VariantVariantConverter<types::TermType> smtrat::parser::TermParser::Converter`

### 0.15.542.2 Constructor & Destructor Documentation

**0.15.542.2.1 `TermParser()`** `smtrat::parser::TermParser::TermParser (`  
`Theories * theories ) [inline]`

### 0.15.542.3 Field Documentation

**0.15.542.3.1 `attribute`** `AttributeParser smtrat::parser::TermParser::attribute`

**0.15.542.3.2 binding** `qi::rule<Iterator, Skipper> smtrat::parser::TermParser::binding`

**0.15.542.3.3 converter** `Converter smtrat::parser::TermParser::converter`

**0.15.542.3.4 main** `qi::rule<Iterator, types::TermType(), Skipper> smtrat::parser::TermParser::main`

**0.15.542.3.5 qualifiedidentifier** `QualifiedIdentifierParser smtrat::parser::TermParser::qualifiedidentifier`

**0.15.542.3.6 sortedvariable** `SortedVariableParser smtrat::parser::TermParser::sortedvariable`

**0.15.542.3.7 speccconstant** `SpecConstantParser smtrat::parser::TermParser::speccconstant`

**0.15.542.3.8 symbol** `SymbolParser smtrat::parser::TermParser::symbol`

**0.15.542.3.9 termop** `qi::rule<Iterator, types::TermType(), Skipper> smtrat::parser::TermParser::termop`

**0.15.542.3.10 theories** `Theories* smtrat::parser::TermParser::theories`

## 0.15.543 smtrat::mcsat::vs::helper::TestCandidate Struct Reference

```
#include <VSHelper.h>
```

### Public Member Functions

- `bool operator==(const TestCandidate &other) const`

### Data Fields

- `carl::vs::Term< Poly > term`
- `ConstraintsT side_condition`

### 0.15.543.1 Member Function Documentation

**0.15.543.1.1 operator==()** `bool smtrat::mcsat::vs::helper::TestCandidate::operator== ( const TestCandidate & other ) const [inline]`

### 0.15.543.2 Field Documentation

**0.15.543.2.1 side\_condition** `ConstraintsT smtrat::mcsat::vs::helper::TestCandidate::side_condition`

**0.15.543.2.2 term** carl::vs::Term<[Poly](#)> smtrat::mcsat::vs::helper::TestCandidate::term

## 0.15.544 smtrat::parser::Theories Struct Reference

The [Theories](#) class combines all implemented theories and provides a single interface to interact with all theories at once.

```
#include <Theories.h>
```

### Data Structures

- struct [ConstantAdder](#)  
*Helper functor for [addConstants\(\)](#) method.*
- struct [SimpleSortAdder](#)  
*Helper functor for [addSimpleSorts\(\)](#) method.*

### Public Types

- [typedef boost::mpl::vector< CoreTheory \\*, ArithmeticTheory \\*, BitvectorTheory \\*, UninterpretedTheory \\*, BooleanEncodingTheory \\* >::type Modules](#)

### Public Member Functions

- [Theories \(ParserState \\*state\)](#)
- [~Theories \(\)](#)
- void [addGlobalFormulas \(FormulasT &formulas\)](#)
- void [declareVariable \(const std::string &name, const carl::Sort &sort\)](#)
- void [declareFunction \(const std::string &name, const std::vector< carl::Sort > &args, const carl::Sort &sort\)](#)
- [types::VariableType declareFunctionArgument \(const std::pair< std::string, carl::Sort > &arg\)](#)
- void [defineFunction \(const std::string &name, const std::vector< types::VariableType > &arguments, const carl::Sort &sort, const types::TermType &definition\)](#)
- [types::TermType resolveSymbol \(const Identifier &identifier\) const](#)
- [types::VariableType resolveVariable \(const Identifier &identifier\) const](#)
- void [pushExpressionScope \(std::size\\_t n\)](#)
- void [popExpressionScope \(std::size\\_t n\)](#)
- void [pushScriptScope \(std::size\\_t n\)](#)
- void [popScriptScope \(std::size\\_t n\)](#)
- const auto & [annotateTerm \(const types::TermType &term, const std::vector< Attribute > &attributes\)](#)
- void [handleLet \(const std::string &symbol, const types::TermType &term\)](#)
- [types::TermType handleITE \(const std::vector< types::TermType > &arguments\)](#)
- [types::TermType handleDistinct \(const std::vector< types::TermType > &arguments\)](#)
- bool [instantiate \(const UserFunctionInstantiator &function, const types::VariableType &var, const types::TermType &repl, types::TermType &subject\)](#)
- bool [instantiateUserFunction \(const UserFunctionInstantiator &function, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors\)](#)
- [types::TermType functionCall \(const Identifier &identifier, const std::vector< types::TermType > &arguments\)](#)

### Static Public Member Functions

- static void [addConstants \(qi::symbols< char, types::ConstType > &constants\)](#)  
*Collects constants from all theory modules.*
- static void [addSimpleSorts \(qi::symbols< char, carl::Sort > &sorts\)](#)  
*Collects simple sorts from all theory modules.*

### 0.15.544.1 Detailed Description

The [Theories](#) class combines all implemented theories and provides a single interface to interact with all theories at once.

### 0.15.544.2 Member Typedef Documentation

**0.15.544.2.1 Modules** `typedef boost::mpl::vector< CoreTheory*, ArithmeticTheory*, BitvectorTheory*, UninterpretedTheory*, BooleanEncodingTheory* >::type smrat::parser::Theories::Modules`

### 0.15.544.3 Constructor & Destructor Documentation

**0.15.544.3.1 Theories()** `smrat::parser::Theories::Theories ( ParserState * state ) [inline]`

**0.15.544.3.2 ~Theories()** `smrat::parser::Theories::~Theories ( ) [inline]`

### 0.15.544.4 Member Function Documentation

**0.15.544.4.1 addConstants()** `static void smrat::parser::Theories::addConstants ( qi::symbols< char, types::ConstType > & constants ) [inline], [static]`  
Collects constants from all theory modules.

**0.15.544.4.2 addGlobalFormulas()** `void smrat::parser::Theories::addGlobalFormulas ( FormulasT & formulas ) [inline]`

**0.15.544.4.3 addSimpleSorts()** `static void smrat::parser::Theories::addSimpleSorts ( qi::symbols< char, carl::Sort > & sorts ) [inline], [static]`  
Collects simple sorts from all theory modules.

**0.15.544.4.4 annotateTerm()** `const auto& smrat::parser::Theories::annotateTerm ( const types::TermType & term, const std::vector< Attribute > & attributes ) [inline]`

**0.15.544.4.5 declareFunction()** `void smrat::parser::Theories::declareFunction ( const std::string & name, const std::vector< carl::Sort > & args, const carl::Sort & sort ) [inline]`

**0.15.544.4.6 declareFunctionArgument()** `types::VariableType smrat::parser::Theories::declareFunctionArgument ( const std::pair< std::string, carl::Sort > & arg ) [inline]`

**0.15.544.4.7 declareVariable()** `void smrat::parser::Theories::declareVariable ( const std::string & name, const carl::Sort & sort ) [inline]`

```
0.15.544.4.8 defineFunction() void smtrat::parser::Theories::defineFunction (
 const std::string & name,
 const std::vector< types::VariableType > & arguments,
 const carl::Sort & sort,
 const types::TermType & definition) [inline]
```

**Todo** check that definition matches the sort

```
0.15.544.4.9 functionCall() types::TermType smtrat::parser::Theories::functionCall (
 const Identifier & identifier,
 const std::vector< types::TermType > & arguments) [inline]
```

```
0.15.544.4.10 handleDistinct() types::TermType smtrat::parser::Theories::handleDistinct (
 const std::vector< types::TermType > & arguments) [inline]
```

```
0.15.544.4.11 handleITE() types::TermType smtrat::parser::Theories::handleITE (
 const std::vector< types::TermType > & arguments) [inline]
```

```
0.15.544.4.12 handleLet() void smtrat::parser::Theories::handleLet (
 const std::string & symbol,
 const types::TermType & term) [inline]
```

```
0.15.544.4.13 instantiate() bool smtrat::parser::Theories::instantiate (
 const UserFunctionInstantiator & function,
 const types::VariableType & var,
 const types::TermType & repl,
 types::TermType & subject) [inline]
```

```
0.15.544.4.14 instantiateUserFunction() bool smtrat::parser::Theories::instantiateUserFunction (
 const UserFunctionInstantiator & function,
 const std::vector< types::TermType > & arguments,
 types::TermType & result,
 TheoryError & errors) [inline]
```

```
0.15.544.4.15 popExpressionScope() void smtrat::parser::Theories::popExpressionScope (
 std::size_t n) [inline]
```

```
0.15.544.4.16 popScriptScope() void smtrat::parser::Theories::popScriptScope (
 std::size_t n) [inline]
```

```
0.15.544.4.17 pushExpressionScope() void smtrat::parser::Theories::pushExpressionScope (
 std::size_t n) [inline]
```

```
0.15.544.4.18 pushScriptScope() void smtrat::parser::Theories::pushScriptScope (
 std::size_t n) [inline]
```

**0.15.544.4.19 `resolveSymbol()`** `types::TermType smtrat::parser::Theories::resolveSymbol ( const Identifier & identifier ) const [inline]`

**0.15.544.4.20 `resolveVariable()`** `types::VariableType smtrat::parser::Theories::resolveVariable ( const Identifier & identifier ) const [inline]`

## 0.15.545 `smtrat::parser::TheoryError` Struct Reference

```
#include <Common.h>
```

### Public Member Functions

- `TheoryError & operator()` (`const std::string &theory`)
- `TheoryError & next ()`
- template<typename T>  
`TheoryError & operator<< (const T &t)`

### Friends

- `std::ostream & operator<< (std::ostream &os, const TheoryError &te)`

## 0.15.545.1 Member Function Documentation

**0.15.545.1.1 `next()`** `TheoryError& smtrat::parser::TheoryError::next ( ) [inline]`

**0.15.545.1.2 `operator()`** `TheoryError& smtrat::parser::TheoryError::operator() ( const std::string & theory ) [inline]`

**0.15.545.1.3 `operator<<()`** `template<typename T > TheoryError& smtrat::parser::TheoryError::operator<< ( const T & t ) [inline]`

## 0.15.545.2 Friends And Related Function Documentation

**0.15.545.2.1 `operator<<`** `std::ostream& operator<< ( std::ostream & os, const TheoryError & te ) [friend]`

## 0.15.546 `smtrat::mcsat::TheoryLevel` Struct Reference

```
#include <MCASATMixin.h>
```

### Data Fields

- `carl::Variable variable = carl::Variable::NO_VARIABLE`  
*Theory variable for this level.*
- `Minisat::Lit decisionLiteral = Minisat::lit_Undef`  
*Literal that assigns this theory variable.*
- `std::vector< Minisat::Var > decidedVariables`  
*Boolean variables univariate in this theory variable.*

### 0.15.546.1 Field Documentation

**0.15.546.1.1 decidedVariables** std::vector<[Minisat::Var](#)> smtrat::mcsat::TheoryLevel::decidedVariables  
Boolean variables univariate in this theory variable.

**0.15.546.1.2 decisionLiteral** [Minisat::Lit](#) smtrat::mcsat::TheoryLevel::decisionLiteral = [Minisat::lit\\_Undef](#)  
Literal that assigns this theory variable.

**0.15.546.1.3 variable** carl::Variable smtrat::mcsat::TheoryLevel::variable = carl::Variable::NO\_VARIABLE  
Theory variable for this level.

## 0.15.547 smtrat::Module::TheoryPropagation Struct Reference

```
#include <Module.h>
```

### Public Member Functions

- [TheoryPropagation \(FormulasT &&\\_premise, const FormulaT &\\_conclusion\)](#)  
*Constructor.*
- [TheoryPropagation \(\)=delete](#)
- [TheoryPropagation \(const TheoryPropagation &\)=delete](#)
- [TheoryPropagation \(TheoryPropagation &&\\_toMove\)](#)
- [TheoryPropagation & operator= \(const TheoryPropagation &\\_toMove\)=delete](#)
- [TheoryPropagation & operator= \(TheoryPropagation &&\\_toMove\)](#)
- [~TheoryPropagation \(\)](#)

### Data Fields

- [FormulasT mPremise](#)  
*The constraints under which the propagated constraint holds.*
- [FormulaT mConclusion](#)  
*The propagated constraint.*

### 0.15.547.1 Constructor & Destructor Documentation

**0.15.547.1.1 TheoryPropagation() [1/4]** smtrat::Module::TheoryPropagation::TheoryPropagation (   
 [FormulasT](#) && [\\_premise](#),  
 const [FormulaT](#) & [\\_conclusion](#) ) [inline]  
Constructor.

**0.15.547.1.2 TheoryPropagation() [2/4]** smtrat::Module::TheoryPropagation::TheoryPropagation ( )  
[delete]

**0.15.547.1.3 TheoryPropagation() [3/4]** smtrat::Module::TheoryPropagation::TheoryPropagation (   
 const [TheoryPropagation](#) & ) [delete]

**0.15.547.1.4 TheoryPropagation()** [4/4] smtrat::Module::TheoryPropagation::TheoryPropagation ( TheoryPropagation && \_toMove ) [inline]

**0.15.547.1.5 ~TheoryPropagation()** smtrat::Module::TheoryPropagation::~TheoryPropagation ( ) [inline]

## 0.15.547.2 Member Function Documentation

**0.15.547.2.1 operator=()** [1/2] TheoryPropagation& smtrat::Module::TheoryPropagation::operator= ( const TheoryPropagation & \_toMove ) [delete]

**0.15.547.2.2 operator=()** [2/2] TheoryPropagation& smtrat::Module::TheoryPropagation::operator= ( TheoryPropagation && \_toMove ) [inline]

## 0.15.547.3 Field Documentation

**0.15.547.3.1 mConclusion** FormulaT smtrat::Module::TheoryPropagation::mConclusion  
The propagated constraint.

**0.15.547.3.2 mPremise** FormulasT smtrat::Module::TheoryPropagation::mPremise  
The constraints under which the propagated constraint holds.

## 0.15.548 smtrat::TheoryVarSchedulerStatic< vot > Class Template Reference

Schedules theory variables statically.

```
#include <VarSchedulerMcsat.h>
```

### Public Member Functions

- template<typename BaseModule >  
**TheoryVarSchedulerStatic** (BaseModule &baseModule)
- void **insert** (Minisat::Var var)
- bool **empty** ()
- Minisat::Lit **pop** ()
- void **print** () const
- template<typename Constraints >  
void **rebuildTheoryVars** (const Constraints &c)
- size\_t **level** ()  
*Level of the next theory variable.*
- size\_t **univariateLevel** (Minisat::Var v)  
*Returns the level in which the given constraint is univariate.*
- void **rebuild** ()  
*Rebuild heap.*
- void **increaseActivity** (Minisat::Var)
- void **decreaseActivity** (Minisat::Var)
- void **rebuildActivities** ()
- void **attachClause** (Minisat::CRef)
- void **detachClause** (Minisat::CRef)
- void **relocateClauses** (std::function< void(Minisat::CRef &) >)

## Protected Attributes

- std::function< bool(Minisat::Var)> **isTheoryVar**
- std::function< carl::Variable(Minisat::Var)> **carlVar**
- std::function< Minisat::Var(carl::Variable)> **minisatVar**
- std::function< Model()> **currentModel**
- std::function< double(Minisat::Var)> **getActivity**
- std::function< char(Minisat::Var)> **getPolarity**
- std::function< void(Minisat::Var, bool)> **setPolarity**
- std::function< bool(Minisat::Var)> **isDecisionVar**
- std::function< bool(Minisat::Var)> **isBoolValueUndef**
- std::function< bool(Minisat::Var)> **isTheoryAbstraction**
- std::function< const FormulaT &(Minisat::Var)> **reabstractVariable**
- std::function< const FormulaT &(Minisat::Lit)> **reabstractLiteral**
- std::function< const Minisat::Clause &(Minisat::CRef)> **getClause**
- std::function< Minisat::lbool(Minisat::Var)> **getBoolVarValue**
- std::function< Minisat::lbool(Minisat::Lit)> **getBoolLitValue**
- std::function< unsigned(const FormulaT &)> **currentlySatisfiedByBackend**
- std::function< Minisat::Var(const FormulaT &)> **abstractVariable**
- std::function< const Minisat::Lit(const FormulaT &)> **abstractLiteral**
- std::function< bool(const FormulaT &)> **isAbstractedFormula**

### 0.15.548.1 Detailed Description

```
template<mcsat::VariableOrdering vot>
class smrat::TheoryVarSchedulerStatic< vot >
```

Schedules theory variables statically.  
Should not be used directly.

### 0.15.548.2 Constructor & Destructor Documentation

```
0.15.548.2.1 TheoryVarSchedulerStatic() template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smrat::TheoryVarSchedulerStatic< vot >::TheoryVarSchedulerStatic (
 BaseModule & baseModule) [inline], [explicit]
```

### 0.15.548.3 Member Function Documentation

```
0.15.548.3.1 attachClause() void smrat::VarSchedulerBase::attachClause (
 Minisat::CRef) [inline], [inherited]
```

```
0.15.548.3.2 decreaseActivity() void smrat::VarSchedulerBase::decreaseActivity (
 Minisat::Var) [inline], [inherited]
```

```
0.15.548.3.3 detachClause() void smrat::VarSchedulerBase::detachClause (
 Minisat::CRef) [inline], [inherited]
```

```
0.15.548.3.4 empty() template<mcsat::VariableOrdering vot>
bool smrat::TheoryVarSchedulerStatic< vot >::empty () [inline]
```

**0.15.548.3.5 increaseActivity()** void smrat::VarSchedulerBase::increaseActivity ( Minisat::Var ) [inline], [inherited]

**0.15.548.3.6 insert()** template<mcsat::VariableOrdering vot> void smrat::TheoryVarSchedulerStatic< vot >::insert ( Minisat::Var var ) [inline]

**0.15.548.3.7 level()** template<mcsat::VariableOrdering vot> size\_t smrat::TheoryVarSchedulerStatic< vot >::level () [inline]  
Level of the next theory variable.

**0.15.548.3.8 pop()** template<mcsat::VariableOrdering vot> Minisat::Lit smrat::TheoryVarSchedulerStatic< vot >::pop () [inline]

**0.15.548.3.9 print()** template<mcsat::VariableOrdering vot> void smrat::TheoryVarSchedulerStatic< vot >::print () const [inline]

**0.15.548.3.10 rebuild()** void smrat::VarSchedulerBase::rebuild () [inline], [inherited]  
Rebuild heap.

**0.15.548.3.11 rebuildActivities()** void smrat::VarSchedulerBase::rebuildActivities () [inline], [inherited]

**0.15.548.3.12 rebuildTheoryVars()** template<mcsat::VariableOrdering vot> template<typename Constraints > void smrat::TheoryVarSchedulerStatic< vot >::rebuildTheoryVars ( const Constraints & c ) [inline]

**0.15.548.3.13 relocateClauses()** void smrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &) > ) [inline], [inherited]

**0.15.548.3.14 univariateLevel()** template<mcsat::VariableOrdering vot> size\_t smrat::TheoryVarSchedulerStatic< vot >::univariateLevel ( Minisat::Var v ) [inline]

Returns the level in which the given constraint is univariate.

## 0.15.548.4 Field Documentation

**0.15.548.4.1 abstractLiteral** std::function<const Minisat::Lit (const FormulaT&)> smrat::VarSchedulerBase::abstractLiteral [protected], [inherited]

**0.15.548.4.2 abstractVariable** std::function< Minisat::Var (const FormulaT&)> smrat::VarSchedulerBase::abstractVariable [protected], [inherited]

**0.15.548.4.3 `carlVar`** std::function<carl::Variable([Minisat::Var](#))> smtrat::VarSchedulerMcsatBase::carlVar [protected], [inherited]

**0.15.548.4.4 `currentlySatisfiedByBackend`** std::function<unsigned(const [FormulaT](#)&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.548.4.5 `currentModel`** std::function<[Model](#)()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]

**0.15.548.4.6 `getActivity`** std::function<double([Minisat::Var](#))> smtrat::VarSchedulerBase::getActivity [protected], [inherited]

**0.15.548.4.7 `getBoolLitValue`** std::function<[Minisat::lbool](#)([Minisat::Lit](#))> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

**0.15.548.4.8 `getBoolVarValue`** std::function<[Minisat::lbool](#)([Minisat::Var](#))> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

**0.15.548.4.9 `getClause`** std::function<const [Minisat::Clause](#)&([Minisat::CRef](#))> smtrat::VarSchedulerBase::getClause [protected], [inherited]

**0.15.548.4.10 `getPolarity`** std::function<char([Minisat::Var](#))> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

**0.15.548.4.11 `isAbstractedFormula`** std::function<bool(const [FormulaT](#)&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

**0.15.548.4.12 `isBoolValueUndef`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]

**0.15.548.4.13 `isDecisionVar`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

**0.15.548.4.14 `isTheoryAbstraction`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]

**0.15.548.4.15 `isTheoryVar`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]

**0.15.548.4.16 `minisatVar`** std::function<[Minisat::Var](#)(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]

**0.15.548.4.17 reabstractLiteral** std::function<const [FormulaT](#)& ([Minisat](#)::Lit)> smtrat::VarScheduler<Base>::reabstractLiteral [protected], [inherited]

**0.15.548.4.18 reabstractVariable** std::function<const [FormulaT](#)& ([Minisat](#)::Var)> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]

**0.15.548.4.19 setPolarity** std::function<void([Minisat](#)::Var, bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]

## 0.15.549 smtrat::ThreadPool Class Reference

```
#include <ThreadPool.h>
```

### Public Member Functions

- [ThreadPool](#) (std::size\_t maxThreads)
- [~ThreadPool](#) ()
- [Answer runBackends](#) (const std::vector< [Module](#) \* > &\_modules, bool \_final, bool \_full, carl::Variable \_objective)

### 0.15.549.1 Constructor & Destructor Documentation

**0.15.549.1.1 ThreadPool()** smtrat::ThreadPool::ThreadPool ( std::size\_t maxThreads ) [inline]

**0.15.549.1.2 ~ThreadPool()** smtrat::ThreadPool::~ThreadPool ( ) [inline]

### 0.15.549.2 Member Function Documentation

**0.15.549.2.1 runBackends()** [Answer](#) smtrat::ThreadPool::runBackends ( const std::vector< [Module](#) \* > & \_modules, bool \_final, bool \_full, carl::Variable \_objective )

#### Parameters

|                   |  |
|-------------------|--|
| <u>_modules</u>   |  |
| <u>_final</u>     |  |
| <u>_full</u>      |  |
| <u>_objective</u> |  |

## 0.15.550 smtrat::cad::debug::TikzBasePrinter Class Reference

```
#include <TikzHistoryPrinter.h>
```

## Public Member Functions

- virtual void `addNode` (std::size\_t level, const std::string &parent, const std::string &node, const std::string &data)=0
- virtual void `addEdge` (const std::string &src, const std::string &dst, const std::string &data)=0
- void `step` ()
- virtual void `layout` ()=0
- virtual void `writeTo` (std::ostream &os, int xBase) const =0
- virtual std::size\_t `width` () const =0

## Protected Member Functions

- std::string `printableID` (const std::string &raw, const std::string &prefix, std::map< std::string, std::string > &map) const

## Protected Attributes

- std::size\_t `mStep`

### 0.15.550.1 Member Function Documentation

**0.15.550.1.1 `addEdge()`** virtual void smtrat::cad::debug::TikzBasePrinter::addEdge ( const std::string & src, const std::string & dst, const std::string & data ) [pure virtual]

Implemented in [smtrat::cad::debug::TikzTreePrinter](#), and [smtrat::cad::debug::TikzDAGPrinter](#).

**0.15.550.1.2 `addNode()`** virtual void smtrat::cad::debug::TikzBasePrinter::addNode ( std::size\_t level, const std::string & parent, const std::string & node, const std::string & data ) [pure virtual]

Implemented in [smtrat::cad::debug::TikzTreePrinter](#), and [smtrat::cad::debug::TikzDAGPrinter](#).

**0.15.550.1.3 `layout()`** virtual void smtrat::cad::debug::TikzBasePrinter::layout ( ) [pure virtual]

Implemented in [smtrat::cad::debug::TikzTreePrinter](#), and [smtrat::cad::debug::TikzDAGPrinter](#).

**0.15.550.1.4 `printableID()`** std::string smtrat::cad::debug::TikzBasePrinter::printableID ( const std::string & raw, const std::string & prefix, std::map< std::string, std::string > & map ) const [inline], [protected]

**0.15.550.1.5 `step()`** void smtrat::cad::debug::TikzBasePrinter::step ( ) [inline]

**0.15.550.1.6 `width()`** virtual std::size\_t smtrat::cad::debug::TikzBasePrinter::width ( ) const [pure virtual]

Implemented in [smtrat::cad::debug::TikzTreePrinter](#), and [smtrat::cad::debug::TikzDAGPrinter](#).

**0.15.550.1.7 `writeTo()`** `virtual void smtrat::cad::debug::TikzBasePrinter::writeTo ( std::ostream & os, int xBase ) const [pure virtual]`  
Implemented in [smtrat::cad::debug::TikzTreePrinter](#), and [smtrat::cad::debug::TikzDAGPrinter](#).

## 0.15.550.2 Field Documentation

**0.15.550.2.1 `mStep`** `std::size_t smtrat::cad::debug::TikzBasePrinter::mStep [protected]`

## 0.15.551 smtrat::cad::debug::TikzDAGPrinter Class Reference

#include <TikzHistoryPrinter.h>

### Public Member Functions

- void `addNode` (std::size\_t level, const std::string &, const std::string &node, const std::string &data) override
- void `addEdge` (const std::string &src, const std::string &dst, const std::string &data) override
- void `layout` () override
- void `writeTo` (std::ostream &os, int xBase) const override
- std::size\_t `width` () const override
- void `step` ()

### Protected Member Functions

- std::string `printableID` (const std::string &raw, const std::string &prefix, std::map< std::string, std::string > &map) const

### Protected Attributes

- std::size\_t `mStep`

## 0.15.551.1 Member Function Documentation

**0.15.551.1.1 `addEdge()`** `void smtrat::cad::debug::TikzDAGPrinter::addEdge ( const std::string & src, const std::string & dst, const std::string & data ) [inline], [override], [virtual]`  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

**0.15.551.1.2 `addNode()`** `void smtrat::cad::debug::TikzDAGPrinter::addNode ( std::size_t level, const std::string & , const std::string & node, const std::string & data ) [inline], [override], [virtual]`  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

**0.15.551.1.3 `layout()`** `void smtrat::cad::debug::TikzDAGPrinter::layout ( ) [inline], [override], [virtual]`  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

**0.15.551.1.4 printableID()** std::string smtrat::cad::debug::TikzBasePrinter::printableID ( const std::string & raw, const std::string & prefix, std::map< std::string, std::string > & map ) const [inline], [protected], [inherited]

**0.15.551.1.5 step()** void smtrat::cad::debug::TikzBasePrinter::step () [inline], [inherited]

**0.15.551.1.6 width()** std::size\_t smtrat::cad::debug::TikzDAGPrinter::width () const [inline], [override], [virtual]  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

**0.15.551.1.7 writeTo()** void smtrat::cad::debug::TikzDAGPrinter::writeTo ( std::ostream & os, int xBase ) const [inline], [override], [virtual]  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

## 0.15.551.2 Field Documentation

**0.15.551.2.1 mStep** std::size\_t smtrat::cad::debug::TikzBasePrinter::mStep [protected], [inherited]

## 0.15.552 smtrat::cad::debug::TikzHistoryPrinter Class Reference

```
#include <TikzHistoryPrinter.h>
```

### Public Member Functions

- template<typename Printer >  
void [configure](#) (const std::string &name)
- template<typename ID1 , typename ID2 , typename T >  
void [addNode](#) (const std::string &printer, std::size\_t level, const ID1 &parent, const ID2 &node, const T &data)
- template<typename ID1 , typename ID2 , typename T >  
void [addEdge](#) (const std::string &printer, const ID1 &src, const ID2 &dst, const T &data)
- template<typename L >  
void [addLifting](#) (const L &l)
- template<typename P >  
void [addProjection](#) (const P &p)
- void [step](#) ()
- void [layout](#) ()
- void [writeTo](#) (const std::string &filename) const

### 0.15.552.1 Member Function Documentation

**0.15.552.1.1 addEdge()** template<typename ID1 , typename ID2 , typename T >  
void smtrat::cad::debug::TikzHistoryPrinter::addEdge ( const std::string & printer, const ID1 & src, const ID2 & dst, const T & data ) [inline]

**0.15.552.1.2 addLifting()** template<typename L >  
void smtrat::cad::debug::TikzHistoryPrinter::addLifting (  
    const L & l ) [inline]

**0.15.552.1.3 addNode()** template<typename ID1 , typename ID2 , typename T >  
void smtrat::cad::debug::TikzHistoryPrinter::addNode (  
    const std::string & printer,  
    std::size\_t level,  
    const ID1 & parent,  
    const ID2 & node,  
    const T & data ) [inline]

**0.15.552.1.4 addProjection()** template<typename P >  
void smtrat::cad::debug::TikzHistoryPrinter::addProjection (  
    const P & p ) [inline]

**0.15.552.1.5 configure()** template<typename Printer >  
void smtrat::cad::debug::TikzHistoryPrinter::configure (  
    const std::string & name ) [inline]

**0.15.552.1.6 layout()** void smtrat::cad::debug::TikzHistoryPrinter::layout ( ) [inline]

**0.15.552.1.7 step()** void smtrat::cad::debug::TikzHistoryPrinter::step ( ) [inline]

**0.15.552.1.8 writeTo()** void smtrat::cad::debug::TikzHistoryPrinter::writeTo (  
    const std::string & filename ) const [inline]

## 0.15.553 smtrat::cad::debug::TikzTreePrinter Class Reference

#include <TikzHistoryPrinter.h>

### Public Member Functions

- void **setRoot** (const std::string &node, const std::string &data)
- void **addNode** (std::size\_t level, const std::string &parent, const std::string &node, const std::string &data) override
- void **addEdge** (const std::string &src, const std::string &dst, const std::string &data) override
- void **layout** () override
- void **writeTo** (std::ostream &os, int xBase) const override
- virtual std::size\_t **width** () const override
- void **step** ()

### Protected Member Functions

- std::string **printableID** (const std::string &raw, const std::string &prefix, std::map< std::string, std::string > &map) const

### Protected Attributes

- std::size\_t **mStep**

### 0.15.553.1 Member Function Documentation

**0.15.553.1.1 addEdge()** void smtrat::cad::debug::TikzTreePrinter::addEdge ( const std::string & *src*, const std::string & *dst*, const std::string & *data* ) [inline], [override], [virtual]  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

**0.15.553.1.2 addNode()** void smtrat::cad::debug::TikzTreePrinter::addNode ( std::size\_t *level*, const std::string & *parent*, const std::string & *node*, const std::string & *data* ) [inline], [override], [virtual]  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

**0.15.553.1.3 layout()** void smtrat::cad::debug::TikzTreePrinter::layout ( ) [inline], [override], [virtual]  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

**0.15.553.1.4 printableID()** std::string smtrat::cad::debug::TikzBasePrinter::printableID ( const std::string & *raw*, const std::string & *prefix*, std::map< std::string, std::string > & *map* ) const [inline], [protected], [inherited]

**0.15.553.1.5 setRoot()** void smtrat::cad::debug::TikzTreePrinter::setRoot ( const std::string & *node*, const std::string & *data* ) [inline]

**0.15.553.1.6 step()** void smtrat::cad::debug::TikzBasePrinter::step ( ) [inline], [inherited]

**0.15.553.1.7 width()** virtual std::size\_t smtrat::cad::debug::TikzTreePrinter::width ( ) const [inline], [override], [virtual]  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

**0.15.553.1.8 writeTo()** void smtrat::cad::debug::TikzTreePrinter::writeTo ( std::ostream & *os*, int *xBase* ) const [inline], [override], [virtual]  
Implements [smtrat::cad::debug::TikzBasePrinter](#).

### 0.15.553.2 Field Documentation

**0.15.553.2.1 mStep** std::size\_t smtrat::cad::debug::TikzBasePrinter::mStep [protected], [inherited]

## 0.15.554 benchmax::Tool Class Reference

Base class for any tool.

```
#include <Tool.h>
```

### Public Member Functions

- `Tool ()=delete`
- `Tool (const Tool &)=delete`
- `Tool (Tool &&)=delete`
- `Tool (const fs::path &binary, const std::string &arguments)`  
*Creates a generic tool from a binary and command line arguments.*
- `Tool (const std::string &name, const fs::path &binary, const std::string &arguments)`  
*Creates a named tool from a binary and command line arguments.*
- `virtual ~Tool ()=default`
- `Tool & operator= (const Tool &)=delete`
- `Tool & operator= (Tool &&)=delete`
- `std::string name () const`  
*Common name of this tool.*
- `fs::path binary () const`  
*Full path to the binary.*
- `const std::map< std::string, std::string > & attributes () const`  
*A set of attributes, for example compilation options.*
- `std::vector< std::string > resolveDependencies () const`  
*Get dependencies of binary required to run it (via ldd)*
- `std::size_t attributeHash () const`  
*Hash of the attributes.*
- `virtual std::string getCommandline (const std::string &file) const`  
*Compose commandline for this tool and the given input file.*
- `virtual std::string getCommandline (const std::string &file, const std::string &localBinary) const`  
*Compose commandline for this tool with another binary name and the given input file.*
- `virtual std::optional< std::string > parseCommandline (const std::string &cmdline) const`  
*Compose commandline for this tool and the given input file.*
- `virtual bool canHandle (const fs::path &) const`  
*Checks whether this cool can handle this file type.*
- `virtual void additionalResults (const fs::path &, BenchmarkResult &) const`  
*Recover additional results from the tool output.*

### Protected Attributes

- `std::string mName`  
*(Non-unique) identifier for the tool.*
- `fs::path mBinary`  
*Path to the binary.*
- `std::string mArguments`  
*Command line arguments that should be passed to the binary.*
- `std::map< std::string, std::string > mAttributes`  
*Attributes of the tool obtained by introspection of the binary.*

### Friends

- `bool operator< (const Tool &lhs, const Tool &rhs)`  
*Compare two tools.*

### 0.15.554.1 Detailed Description

Base class for any tool.

A tool represents some executable that can be run with some input file. A tool is responsible for

- deciding it is applicable for a given file extension,
- building a command line to execute it and
- parse additional results from stdout / stderr after it has run.

A tool is not to be copied or moved around but should only exist a single time.

### 0.15.554.2 Constructor & Destructor Documentation

**0.15.554.2.1 Tool()** [1/5] `benchmax::Tool::Tool( ) [delete]`

**0.15.554.2.2 Tool()** [2/5] `benchmax::Tool::Tool( const Tool & ) [delete]`

**0.15.554.2.3 Tool()** [3/5] `benchmax::Tool::Tool( Tool && ) [delete]`

**0.15.554.2.4 Tool()** [4/5] `benchmax::Tool::Tool( const fs::path & binary, const std::string & arguments ) [inline]`

Creates a generic tool from a binary and command line arguments.

**0.15.554.2.5 Tool()** [5/5] `benchmax::Tool::Tool( const std::string & name, const fs::path & binary, const std::string & arguments ) [inline]`

Creates a named tool from a binary and command line arguments.

**0.15.554.2.6 ~Tool()** `virtual benchmax::Tool::~Tool( ) [virtual], [default]`

### 0.15.554.3 Member Function Documentation

**0.15.554.3.1 additionalResults()** `virtual void benchmax::Tool::additionalResults( const fs::path &, BenchmarkResult & ) const [inline], [virtual]`

Recover additional results from the tool output.

Reimplemented in [benchmax::Z3](#), [benchmax::SMTRAT\\_Analyzer](#), [benchmax::SMTRAT](#), [benchmax::Minisatp](#), [benchmax::Minisat](#), and [benchmax::MathSAT](#).

**0.15.554.3.2 attributeHash()** `std::size_t benchmax::Tool::attributeHash( ) const [inline]`

Hash of the attributes.

**0.15.554.3.3 `attributes()`** `const std::map<std::string, std::string>& benchmax::Tool::attributes( ) const [inline]`  
A set of attributes, for example compilation options.

**0.15.554.3.4 `binary()`** `fs::path benchmax::Tool::binary( ) const [inline]`  
Full path to the binary.

**0.15.554.3.5 `canHandle()`** `virtual bool benchmax::Tool::canHandle( const fs::path & ) const [inline], [virtual]`

Checks whether this cool can handle this file type.

Reimplemented in [benchmax::Z3](#), [benchmax::SMTRAT\\_OPB](#), [benchmax::SMTRAT\\_Analyzer](#), [benchmax::SMTRAT](#), [benchmax::Minisatp](#), [benchmax::Minisat](#), and [benchmax::MathSAT](#).

**0.15.554.3.6 `getCommandLine()` [1/2]** `virtual std::string benchmax::Tool::getCommandLine( const std::string & file ) const [inline], [virtual]`

Compose commandline for this tool and the given input file.

**0.15.554.3.7 `getCommandLine()` [2/2]** `virtual std::string benchmax::Tool::getCommandLine( const std::string & file, const std::string & localBinary ) const [inline], [virtual]`

Compose commandline for this tool with another binary name and the given input file.

**0.15.554.3.8 `name()`** `std::string benchmax::Tool::name( ) const [inline]`  
Common name of this tool.

**0.15.554.3.9 `operator=()` [1/2]** `Tool& benchmax::Tool::operator=( const Tool & ) [delete]`

**0.15.554.3.10 `operator=()` [2/2]** `Tool& benchmax::Tool::operator=( Tool && ) [delete]`

**0.15.554.3.11 `parseCommandLine()`** `virtual std::optional<std::string> benchmax::Tool::parseCommandline( const std::string & cmdline ) const [inline], [virtual]`  
Compose commandline for this tool and the given input file.

**0.15.554.3.12 `resolveDependencies()`** `std::vector<std::string> benchmax::Tool::resolveDependencies( ) const [inline]`  
Get dependencies of binary required to run it (via ldd)

#### 0.15.554.4 Friends And Related Function Documentation

```
0.15.554.4.1 operator< bool operator< (
 const Tool & lhs,
 const Tool & rhs) [friend]
```

Compare two tools.

#### 0.15.554.5 Field Documentation

**0.15.554.5.1 mArguments** std::string benchmax::Tool::mArguments [protected]  
Command line arguments that should be passed to the binary.

**0.15.554.5.2 mAttributes** std::map<std::string, std::string> benchmax::Tool::mAttributes [protected]  
Attributes of the tool obtained by introspection of the binary.

**0.15.554.5.3 mBinary** fs::path benchmax::Tool::mBinary [protected]  
Path to the binary.

**0.15.554.5.4 mName** std::string benchmax::Tool::mName [protected]  
(Non-unique) identifier for the tool.

### 0.15.555 benchmax::settings::ToolSettings Struct Reference

Tool-related settings.

```
#include <Tools.h>
```

#### Data Fields

- bool **collect\_statistics**  
*Whether or not to collect statistics.*
- std::vector< std::filesystem::path > **tools\_generic**  
*Generic tools.*
- std::vector< std::filesystem::path > **tools\_mathsat**  
*MathSAT with SMT-LIB interface.*
- std::vector< std::filesystem::path > **tools\_minisat**  
*Minisat with DIMACS interface.*
- std::vector< std::filesystem::path > **tools\_minisatp**  
*Minisatp with OPB interface.*
- std::vector< std::filesystem::path > **tools\_smrat**  
*SMT-RAT with SMT-LIB interface.*
- std::vector< std::filesystem::path > **tools\_smrat\_opb**  
*SMT-RAT with OPB interface.*
- std::vector< std::filesystem::path > **tools\_smrat\_analyzer**  
*SMT-RAT formula analyzer.*
- std::vector< std::filesystem::path > **tools\_z3**  
*z3 with SMT-LIB interface.*
- std::filesystem::path **tools\_common\_prefix**  
*Common prefix of tool binaries to simplify output.*

#### 0.15.555.1 Detailed Description

Tool-related settings.

### 0.15.555.2 Field Documentation

**0.15.555.2.1 collect\_statistics** bool benchmax::settings::ToolSettings::collect\_statistics  
Whether or not to collect statistics.

**0.15.555.2.2 tools\_common\_prefix** std::filesystem::path benchmax::settings::ToolSettings::tools\_common\_prefix  
Common prefix of tool binaries to simplify output.

**0.15.555.2.3 tools\_generic** std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools\_generic  
Generic tools.

**0.15.555.2.4 tools\_mathsat** std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools\_mathsat  
[MathSAT](#) with SMT-LIB interface.

**0.15.555.2.5 tools\_minisat** std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools\_minisat  
[Minisat](#) with DIMACS interface.

**0.15.555.2.6 tools\_minisatp** std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools\_minisatp  
[Minisatp](#) with OPB interface.

**0.15.555.2.7 tools\_smrat** std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools\_smrat  
SMT-RAT with SMT-LIB interface.

**0.15.555.2.8 tools\_smrat\_analyzer** std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools\_smrat\_analyzer  
SMT-RAT formula analyzer.

**0.15.555.2.9 tools\_smrat\_opb** std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools\_smrat\_opb  
SMT-RAT with OPB interface.

**0.15.555.2.10 tools\_z3** std::vector<std::filesystem::path> benchmax::settings::ToolSettings::tools\_z3  
z3 with SMT-LIB interface.

## 0.15.556 smrat::TotalizerEncoder Class Reference

[TotalizerEncoder](#) implements the Totalizer encoding as described in "Incremental Cardinality Constraints for Max-SAT" by Martins et al [https://doi.org/10.1007/978-3-319-10428-7\\_39](https://doi.org/10.1007/978-3-319-10428-7_39).  
#include <TotalizerEncoder.h>

## Public Member Functions

- `TotalizerEncoder ()`
- `~TotalizerEncoder ()`
- `bool canEncode (const ConstraintT &constraint)`
- `Rational encodingSize (const ConstraintT &constraint)`
- `std::string name ()`
- `std::optional< FormulaT > encode (const ConstraintT &constraint)`

*Encodes an arbitrary constraint.*

## Data Fields

- `std::size_t problem_size`

## Protected Member Functions

- `std::optional< FormulaT > doEncode (const ConstraintT &constraint)`
- `FormulaT generateVarChain (const std::set< carl::Variable > &vars, carl::FormulaType type)`

### 0.15.556.1 Detailed Description

`TotalizerEncoder` implements the Totalizer encoding as described in "Incremental Cardinality Constraints for Max-SAT" by Martins et al [https://doi.org/10.1007/978-3-319-10428-7\\_39](https://doi.org/10.1007/978-3-319-10428-7_39).

### 0.15.556.2 Constructor & Destructor Documentation

#### 0.15.556.2.1 `TotalizerEncoder()` `smtrat::TotalizerEncoder::TotalizerEncoder () [inline]`

#### 0.15.556.2.2 `~TotalizerEncoder()` `smtrat::TotalizerEncoder::~TotalizerEncoder ()`

### 0.15.556.3 Member Function Documentation

#### 0.15.556.3.1 `canEncode()` `bool smtrat::TotalizerEncoder::canEncode (const ConstraintT & constraint) [virtual]`

Implements `smtrat::PseudoBoolEncoder`.

#### 0.15.556.3.2 `doEncode()` `std::optional< FormulaT > smtrat::TotalizerEncoder::doEncode (const ConstraintT & constraint) [protected], [virtual]`

Implements `smtrat::PseudoBoolEncoder`.

#### 0.15.556.3.3 `encode()` `std::optional< FormulaT > smtrat::PseudoBoolEncoder::encode (const ConstraintT & constraint) [inherited]`

Encodes an arbitrary constraint.

#### Returns

encoded formula

**0.15.556.3.4 encodingSize()** `Rational smtrat::TotalizerEncoder::encodingSize ( const ConstraintT & constraint ) [virtual]`  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

**0.15.556.3.5 generateVarChain()** `FormulaT smtrat::PseudoBoolEncoder::generateVarChain ( const std::set< carl::Variable > & vars, carl::FormulaType type ) [protected], [inherited]`

**0.15.556.3.6 name()** `std::string smtrat::TotalizerEncoder::name () [inline], [virtual]`  
Reimplemented from [smtrat::PseudoBoolEncoder](#).

#### 0.15.556.4 Field Documentation

**0.15.556.4.1 problem\_size** `std::size_t smtrat::PseudoBoolEncoder::problem_size [inherited]`

### 0.15.557 smtrat::TotalizerTree Class Reference

```
#include <TotalizerEncoder.h>
```

#### Public Member Functions

- `TotalizerTree (const std::set< carl::Variable > &variables)`
- `~TotalizerTree ()`
- `bool isLeaf ()`
- `TotalizerTree & left ()`
- `TotalizerTree & right ()`
- `std::vector< carl::Variable > variables ()`

#### 0.15.557.1 Constructor & Destructor Documentation

**0.15.557.1.1 TotalizerTree()** `smtrat::TotalizerTree::TotalizerTree ( const std::set< carl::Variable > & variables )`

**0.15.557.1.2 ~TotalizerTree()** `smtrat::TotalizerTree::~TotalizerTree () [inline]`

#### 0.15.557.2 Member Function Documentation

**0.15.557.2.1 isLeaf()** `bool smtrat::TotalizerTree::isLeaf () [inline]`

**0.15.557.2.2 left()** `TotalizerTree& smtrat::TotalizerTree::left () [inline]`

**0.15.557.2.3 right()** `TotalizerTree& smtrat::TotalizerTree::right () [inline]`

**0.15.557.2.4 variables()** `std::vector<carl::Variable> smtrat::TotalizerTree::variables () [inline]`

### 0.15.558 smtrat::cad::projection\_compare::type Struct Reference

```
#include <ProjectionComparator.h>
```

### 0.15.559 smtrat::cad::sample\_compare::type Struct Reference

```
#include <SampleComparator.h>
```

## 0.15.560 smtrat::uf\_impl::uf\_data\_wrapper< T, compression > Class Template Reference

Wrapper around data entry and [uf\\_rank\\_and\\_class](#); this class contains a value of type T and both rank and class of an entry in the UF datastructure.

```
#include <union_find.h>
```

### Public Types

- `typedef T value_type`

### Public Member Functions

- `uf_data_wrapper (T value_, std::size_t class_index)`
- `T & value () noexcept`
- `const T & value () const noexcept`
- `std::size_t rank () const noexcept`
- `std::size_t class_ () const noexcept`
- `void set_class (std::size_t newclass) noexcept`
- `void increment_rank () noexcept`
- `void set_rank (std::size_t rank_) noexcept`

### 0.15.560.1 Detailed Description

```
template<typename T, bool compression>
class smtrat::uf_impl::uf_data_wrapper< T, compression >
```

Wrapper around data entry and [uf\\_rank\\_and\\_class](#); this class contains a value of type T and both rank and class of an entry in the UF datastructure.

### 0.15.560.2 Member Typedef Documentation

```
0.15.560.2.1 value_type template<typename T , bool compression>
typedef T smtrat::uf_impl::uf_data_wrapper< T, compression >::value_type
```

### 0.15.560.3 Constructor & Destructor Documentation

```
0.15.560.3.1 uf_data_wrapper() template<typename T , bool compression>
smtrat::uf_impl::uf_data_wrapper< T, compression >::uf_data_wrapper (
 T value_,
 std::size_t class_index) [inline]
```

### 0.15.560.4 Member Function Documentation

```
0.15.560.4.1 class_() template<typename T , bool compression>
std::size_t smtrat::uf_impl::uf_data_wrapper< T, compression >::class_ () const [inline],
[noexcept]
```

```
0.15.560.4.2 increment_rank() template<typename T , bool compression>
void smtrat::uf_impl::uf_data_wrapper< T, compression >::increment_rank () [inline], [noexcept]
```

```
0.15.560.4.3 rank() template<typename T , bool compression>
std::size_t smtrat::uf_impl::uf_data_wrapper< T, compression >::rank () const [inline],
[noexcept]
```

```
0.15.560.4.4 set_class() template<typename T , bool compression>
void smtrat::uf_impl::uf_data_wrapper< T, compression >::set_class (
 std::size_t newclass) [inline], [noexcept]
```

```
0.15.560.4.5 set_rank() template<typename T , bool compression>
void smtrat::uf_impl::uf_data_wrapper< T, compression >::set_rank (
 std::size_t rank_) [inline], [noexcept]
```

```
0.15.560.4.6 value() [1/2] template<typename T , bool compression>
const T& smtrat::uf_impl::uf_data_wrapper< T, compression >::value () const [inline], [noexcept]
```

```
0.15.560.4.7 value() [2/2] template<typename T , bool compression>
T& smtrat::uf_impl::uf_data_wrapper< T, compression >::value () [inline], [noexcept]
```

## 0.15.561 smtrat::uf\_impl::uf\_data\_wrapper< void, compression > Class Template Reference

Special case of the [uf\\_data\\_wrapper](#) in case the data type is void.

```
#include <union_find.h>
```

### Public Member Functions

- [uf\\_data\\_wrapper](#) (std::size\_t class\_index)
- std::size\_t [rank](#) () const noexcept
- std::size\_t [class\\_](#) () const noexcept
- void [set\\_class](#) (std::size\_t newclass) noexcept
- void [increment\\_rank](#) () noexcept
- void [set\\_rank](#) (std::size\_t rank\_) noexcept

### 0.15.561.1 Detailed Description

```
template<bool compression>
class smtrat::uf_impl::uf_data_wrapper< void, compression >
```

Special case of the [uf\\_data\\_wrapper](#) in case the data type is void.

Then, there is no value member and no corresponding member functions.

### 0.15.561.2 Constructor & Destructor Documentation

```
0.15.561.2.1 uf_data_wrapper() template<bool compression>
smtrat::uf_impl::uf_data_wrapper< void, compression >::uf_data_wrapper (
 std::size_t class_index) [inline]
```

### 0.15.561.3 Member Function Documentation

```
0.15.561.3.1 class_() template<bool compression>
std::size_t smtrat::uf_impl::uf_data_wrapper< void, compression >::class_ () const [inline],
[noexcept]
```

```
0.15.561.3.2 increment_rank() template<bool compression>
void smtrat::uf_impl::uf_data_wrapper< void, compression >::increment_rank () [inline],
[noexcept]
```

```
0.15.561.3.3 rank() template<bool compression>
std::size_t smtrat::uf_impl::uf_data_wrapper< void, compression >::rank () const [inline],
[noexcept]
```

```
0.15.561.3.4 set_class() template<bool compression>
void smtrat::uf_impl::uf_data_wrapper< void, compression >::set_class (
 std::size_t newclass) [inline], [noexcept]
```

```
0.15.561.3.5 set_rank() template<bool compression>
void smtrat::uf_impl::uf_data_wrapper< void, compression >::set_rank (
 std::size_t rank_) [inline], [noexcept]
```

## 0.15.562 smtrat::uf\_impl::uf\_rank\_and\_class< compression > Class Template Reference

Wrapper class that contains the rank and parent class of some entry of the UF datastructure.

```
#include <union_find.h>
```

### Public Member Functions

- **uf\_rank\_and\_class** (std::size\_t class\_index)
- std::size\_t **rank** () const noexcept
- std::size\_t **class\_** () const noexcept
- void **set\_class** (std::size\_t newclass) noexcept
- void **increment\_rank** () noexcept
- void **set\_rank** (std::size\_t **rank**)

### 0.15.562.1 Detailed Description

```
template<bool compression>
class smtrat::uf_impl::uf_rank_and_class< compression >
```

Wrapper class that contains the rank and parent class of some entry of the UF datastructure.

The compression parameter (default to off, this seems to be faster, can only enabled if sizeof(std::size\_t) >= 8) controls whether the rank member is compressed together with the parent class. If this is true, sizeof(uf\_rank\_and\_class<true>) is just sizeof(std::size\_t), and the most significant byte of the class is interpreted as rank of the entry.

### 0.15.562.2 Constructor & Destructor Documentation

```
0.15.562.2.1 uf_rank_and_class() template<bool compression>
smtrat::uf_impl::uf_rank_and_class< compression >::uf_rank_and_class (
 std::size_t class_index) [inline]
```

### 0.15.562.3 Member Function Documentation

```
0.15.562.3.1 class_() template<bool compression>
std::size_t smtrat::uf_impl::uf_rank_and_class< compression >::class_ () const [inline], [noexcept]
```

```
0.15.562.3.2 increment_rank() template<bool compression>
void smtrat::uf_impl::uf_rank_and_class< compression >::increment_rank () [inline], [noexcept]
```

```
0.15.562.3.3 rank() template<bool compression>
std::size_t smtrat::uf_impl::uf_rank_and_class< compression >::rank () const [inline], [noexcept]
```

```
0.15.562.3.4 set_class() template<bool compression>
void smtrat::uf_impl::uf_rank_and_class< compression >::set_class (
 std::size_t newclass) [inline], [noexcept]
```

```
0.15.562.3.5 set_rank() template<bool compression>
void smtrat::uf_impl::uf_rank_and_class< compression >::set_rank (
 std::size_t rank) [inline]
```

## 0.15.563 smtrat::uf\_impl::uf\_rank\_and\_class< true > Class Reference

Compressed implementation of [uf\\_rank\\_and\\_class](#).

```
#include <union_find.h>
```

### Public Member Functions

- [uf\\_rank\\_and\\_class](#) (std::size\_t class\_index)
- std::size\_t [rank](#) () const noexcept
- std::size\_t [class\\_](#) () const noexcept
- void [set\\_class](#) (std::size\_t newclass) noexcept
- void [increment\\_rank](#) () noexcept
- void [set\\_rank](#) (std::size\_t rank)

### 0.15.563.1 Detailed Description

Compressed implementation of [uf\\_rank\\_and\\_class](#).

### 0.15.563.2 Constructor & Destructor Documentation

```
0.15.563.2.1 uf_rank_and_class() smtrat::uf_impl::uf_rank_and_class< true >::uf_rank_and_class
(
 std::size_t class_index) [inline]
```

### 0.15.563.3 Member Function Documentation

```
0.15.563.3.1 class_() std::size_t smtrat::uf_impl::uf_rank_and_class< true >::class_ () const
[inline], [noexcept]
```

```
0.15.563.3.2 increment_rank() void smtrat::uf_impl::uf_rank_and_class< true >::increment_rank
() [inline], [noexcept]
```

```
0.15.563.3.3 rank() std::size_t smtrat::uf_impl::uf_rank_and_class< true >::rank () const
[inline], [noexcept]
```

```
0.15.563.3.4 set_class() void smtrat::uf_impl::uf_rank_and_class< true >::set_class (
 std::size_t newclass) [inline], [noexcept]
```

```
0.15.563.3.5 set_rank() void smtrat::uf_impl::uf_rank_and_class< true >::set_rank (
 std::size_t rank) [inline]
```

## 0.15.564 smtrat::UFCegarModule< Settings > Class Template Reference

```
#include <UFCegarModule.h>
```

### Public Types

- **typedef Settings SettingsType**
- **enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }**

### Public Member Functions

- **std::string moduleName () const**
- **UFCegarModule (const ModuleInput \*\_formula, Conditionals &\_conditionals, Manager \*\_manager=nullptr)**
- **~UFCegarModule ()**
- **bool addCore (ModuleInput::const\_iterator \_subformula)**  
*The module has to take the given sub-formula of the received formula into account.*
- **void removeCore (ModuleInput::const\_iterator \_subformula)**  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
- **void updateModel () const**  
*Updates the current assignment into the model.*
- **Answer checkCore ()**  
*Checks the received formula for consistency.*
- **bool inform (const FormulaT &\_constraint)**  
*Informs the module about the given constraint.*
- **void deform (const FormulaT &\_constraint)**  
*The inverse of informing about a constraint.*
- **virtual void init ()**  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*

- bool `add (ModuleInput::const_iterator _subformula)`

*The module has to take the given sub-formula of the received formula into account.*
- virtual `Answer check (bool _final=false, bool _full=true, carl::Variable _objective=carl::Variable::NO_VARIABLE)`

*Checks the received formula for consistency.*
- virtual void `remove (ModuleInput::const_iterator _subformula)`

*Removes everything related to the given sub-formula of the received formula.*
- virtual void `updateAllModels ()`

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
- virtual std::list< std::vector< carl::Variable > > `getModelEqualities () const`

*Partition the variables from the current model into equivalence classes according to their assigned value.*
- unsigned `currentlySatisfiedByBackend (const FormulaT &_formula) const`
- virtual unsigned `currentlySatisfied (const FormulaT &) const`
- bool `receivedVariable (carl::Variable::Arg _var) const`
- `Answer solverState () const`
- std::size\_t `id () const`
- void `setId (std::size_t _id)`

*Sets this modules unique ID to identify itself.*
- `thread_priority threadPriority () const`
- void `setThreadPriority (thread_priority _threadPriority)`

*Sets the priority of this module to get a thread for running its check procedure.*
- const `ModuleInput * pReceivedFormula () const`
- const `ModuleInput & rReceivedFormula () const`
- const `ModuleInput * pPassedFormula () const`
- const `ModuleInput & rPassedFormula () const`
- const `Model & model () const`
- const std::vector< Model > & `allModels () const`
- const std::vector< FormulaSetT > & `infeasibleSubsets () const`
- const std::vector< Module \* > & `usedBackends () const`
- const carl::FastSet< FormulaT > & `constraintsToInform () const`
- const carl::FastSet< FormulaT > & `informedConstraints () const`
- void `addLemma (const FormulaT &_lemma, const LemmaType &_lt=LemmaType::NORMAL, const FormulaT &_preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const std::vector< Lemma > & `lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- carl::Variable `objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`

- Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins` (const `FormulaT` & `_formula`, `FormulaSetT` & `_origins`) const
  - bool `isValidInfeasibleSubset` () const
  - void `checkInfSubsetForMinimality` (`std::vector<FormulaSetT>`::const\_iterator `_infsSubset`, const `std::string` & `_filename`="smaller\_muses", unsigned `_maxSizeDifference`=1) const
- Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual `std::pair<bool, FormulaT>` `getReceivedFormulaSimplified` ()
  - void `print` (const `std::string` & `_initiation`="\*\*\*") const
- Prints everything relevant of the solver.*
- void `printReceivedFormula` (const `std::string` & `_initiation`="\*\*\*") const
- Prints the vector of the received formula.*
- void `printPassedFormula` (const `std::string` & `_initiation`="\*\*\*") const
- Prints the vector of passed formula.*
- void `printInfeasibleSubsets` (const `std::string` & `_initiation`="\*\*\*") const
- Prints the infeasible subsets.*
- void `printModel` (`std::ostream` & `_out`=`std::cout`) const
- Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels` (`std::ostream` & `_out`=`std::cout`)
- Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable` (const `FormulaT` & `_splittingVariable`)

## Static Public Attributes

- static `size_t` `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static `std::vector<Branching>` `mLastBranches` = `std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`  
*Stores the last branches in a cycle buffer.*
- static `size_t` `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static `std::vector<FormulaT>` `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const `FormulaT` & `_constraint`)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const `FormulaT` & `_constraint`)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*

- `ModuleInput::iterator passedFormulaBegin ()`
- `ModuleInput::iterator passedFormulaEnd ()`
- `void addOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- `const FormulaT & getOrigins (ModuleInput::const_iterator _formula) const`

*Gets the origins of the passed formula at the given position.*
- `void getOrigins (const FormulaT &_formula, FormulasT &_origins) const`
- `void getOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `std::pair< ModuleInput::iterator, bool > removeOrigin (ModuleInput::iterator _formula, const FormulaT &_origin)`
- `std::pair< ModuleInput::iterator, bool > removeOrigins (ModuleInput::iterator _formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`
- `void informBackends (const FormulaT &_constraint)`

*Informs all backends of this module about the given constraint.*
- `virtual void addConstraintToInform (const FormulaT &_constraint)`

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- `std::pair< ModuleInput::iterator, bool > addReceivedSubformulaToPassedFormula (ModuleInput::const_iterator _subformula)`

*Copies the given sub-formula of the received formula to the passed formula.*
- `bool originInReceivedFormula (const FormulaT &_origin) const`
- `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula)`

*Adds the given formula to the passed formula with no origin.*
- `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const std::shared_ptr< std::vector< FormulaT >> &_origins)`

*Adds the given formula to the passed formula.*
- `std::pair< ModuleInput::iterator, bool > addSubformulaToPassedFormula (const FormulaT &_formula, const FormulaT &_origin)`

*Adds the given formula to the passed formula.*
- `void generateTrivialInfeasibleSubset ()`

*Stores the trivial infeasible subset being the set of received formulas.*
- `void receivedFormulasAsInfeasibleSubset (ModuleInput::const_iterator _subformula)`

*Stores an infeasible subset consisting only of the given received formula.*
- `std::vector< FormulaT >::const_iterator findBestOrigin (const std::vector< FormulaT > &_origins) const`
- `void getInfeasibleSubsets ()`

*Copies the infeasible subsets of the passed formula.*
- `std::vector< FormulaSetT > getInfeasibleSubsets (const Module &backend) const`

*Get the infeasible subsets the given backend provides.*
- `const Model & backendsModel () const`

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- `void getBackendsModel () const`
- `void getBackendsAllModels () const`

*Stores all models of a backend in the list of all models of this module.*
- `virtual Answer runBackends (bool _final, bool _full, carl::Variable _objective)`

*Runs the backend solvers on the passed formula.*
- `virtual Answer runBackends ()`
- `virtual ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- `void clearPassedFormula ()`
- `std::vector< FormulaT > merge (const std::vector< FormulaT > &, const std::vector< FormulaT > &) const`

*Merges the two vectors of sets into the first one.*
- `size_t determine_smallest_origin (const std::vector< FormulaT > &origins) const`

- bool `probablyLooping` (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const  
*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< FormulaT > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)  
*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &&\_premise=std::vector< FormulaT >())
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, std::vector< FormulaT > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< FormulaT > &\_premise=std::vector< FormulaT >())
- void `splitUnequalConstraint` (const FormulaT &)  
*Adds the following lemmas for the given constraint  $p \neq 0$ .*
- unsigned `checkModel` () const
- void `addInformationRelevantFormula` (const FormulaT &formula)  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< FormulaT > & `getInformationRelevantFormulas` ()  
*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel` (LemmaLevel level)  
*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const Model &\_modelA, const Model &\_modelB)  
*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- std::vector< FormulaSetT > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- Manager \*const `mpManager`  
*A reference to the manager.*
- Model `mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< Model > `mAllModels`  
*Stores all satisfying assignments.*
- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< TheoryPropagation > `mTheoryPropagations`
- std::atomic< Answer > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- **Conditionals mFoundAnswer**

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- std::vector< Module \* > mUsedBackends

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- std::vector< Module \* > mAllBackends

*The backends of this module which have been used.*

- std::vector< Lemma > mLemmas

*Stores the lemmas being valid formulas this module or its backends made.*

- ModuleInput::iterator mFirstSubformulaToPass

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- carl::FastSet< FormulaT > mConstraintsToInform

*Stores the constraints which the backends must be informed about.*

- carl::FastSet< FormulaT > mInformedConstraints

*Stores the position of the first constraint of which no backend has been informed about.*

- ModuleInput::const\_iterator mFirstUncheckedReceivedSubformula

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- unsigned mSmallerMusesCheckCounter

*Counter used for the generation of the smt2 files to check for smaller muses.*

- std::vector< std::size\_t > mVariableCounters

*Maps variables to the number of their occurrences.*

#### 0.15.564.1 Member Typedef Documentation

**0.15.564.1.1 SettingsType** template<typename Settings >  
typedef smtrat::UFCegarModule< Settings >::SettingsType

#### 0.15.564.2 Member Enumeration Documentation

**0.15.564.2.1 LemmaType** enum smtrat::Module::LemmaType : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

#### 0.15.564.3 Constructor & Destructor Documentation

**0.15.564.3.1 UFCegarModule()** template<typename Settings >  
smtrat::UFCegarModule< Settings >::UFCegarModule (

```
 const ModuleInput * _formula,
 Conditionals & _conditionals,
 Manager * _manager = nullptr)
```

**0.15.564.3.2 ~UFCegarModule()** template<typename Settings >  
smtrat::UFCegarModule< Settings >::~UFCegarModule ( )

#### 0.15.564.4 Member Function Documentation

**0.15.564.4.1 add()** bool smtrat::Module::add (   
    ModuleInput::const\_iterator \_subformula ) [inherited]

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.564.4.2 addConstraintToInform()** void smtrat::Module::addConstraintToInform (   
    const FormulaT & \_constraint ) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

##### Parameters

|             |                        |
|-------------|------------------------|
| _constraint | The constraint to add. |
|-------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.564.4.3 addCore()** template<typename Settings >  
bool smtrat::UFCegarModule< Settings >::addCore (   
    ModuleInput::const\_iterator \_subformula ) [virtual]

The module has to take the given sub-formula of the received formula into account.

##### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub-formula to take additionally into account. |
|-------------|----------------------------------------------------|

##### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.564.4.4 addInformationRelevantFormula()** void smtrat::Module::addInformationRelevantFormula (   
    const FormulaT & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

##### Parameters

|         |                |
|---------|----------------|
| formula | Formula to add |
|---------|----------------|

```
0.15.564.4.5 addLemma() void smtrat::Module::addLemma (
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

#### Parameters

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

```
0.15.564.4.6 addOrigin() void smtrat::Module::addOrigin (
```

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

```
0.15.564.4.7 addReceivedSubformulaToPassedFormula() std::pair<ModuleInput::iterator,bool>
smtrat::Module::addReceivedSubformulaToPassedFormula (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>_subformula</i> | The sub-formula of the received formula to copy. |
|--------------------|--------------------------------------------------|

```
0.15.564.4.8 addSubformulaToPassedFormula() [1/3] std::pair<ModuleInput::iterator,bool> smtrat<-
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula. |
|-----------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.564.4.9 addSubformulaToPassedFormula()** [2/3] std::pair<ModuleInput::iterator,bool> smrat::Module::addSubformulaToPassedFormula (

```
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                 |                                                                                                                           |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                 |
| <i>_origin</i>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.564.4.10 addSubformulaToPassedFormula()** [3/3] std::pair<ModuleInput::iterator,bool> smrat::Module::addSubformulaToPassedFormula (

```
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                 |                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula to add to the passed formula.                                                                                               |
| <i>_origins</i> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.564.4.11 allModels()** const std::vector<Model>& smrat::Module::allModels () const [inline], [inherited]

**Returns**

All satisfying assignments, if existent.

**0.15.564.4.12 `anAnswerFound()`** `bool smrat::Module::anAnswerFound ( ) const [inline], [protected], [inherited]`

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

**0.15.564.4.13 `answerFound()`** `const smrat::Conditionals& smrat::Module::answerFound ( ) const [inline], [inherited]`

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.564.4.14 `backendsModel()`** `const Model & smrat::Module::backendsModel ( ) const [protected], [inherited]`

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.564.4.15 `branchAt() [1/4]`** `bool smrat::Module::branchAt ( carl::Variable::Arg _var, const Rational & _value, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]`

**0.15.564.4.16 `branchAt() [2/4]`** `bool smrat::Module::branchAt ( carl::Variable::Arg _var, const Rational & _value, std::vector< FormulaT > && _premise, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]`

**0.15.564.4.17 `branchAt() [3/4]`** `bool smrat::Module::branchAt ( const Poly & _polynomial, bool _integral, const Rational & _value, bool _leftCaseWeak = true, bool _preferLeftCase = true, bool _useReceivedFormulaAsPremise = false, const std::vector< FormulaT > & _premise = std::vector<FormulaT>() ) [inline], [protected], [inherited]`

**0.15.564.4.18 `branchAt() [4/4]`** `bool smrat::Module::branchAt ( const Poly & _polynomial, bool _integral, const Rational & _value,`

```
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

#### 0.15.564.4.19 `check()`

```
Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

#### 0.15.564.4.20 `checkCore()`

```
template<typename Settings >
Answer smtrat::UFCegarModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.564.4.21 checkInfSubsetForMinimality() void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

```
0.15.564.4.22 checkModel() unsigned smtrat::Module::checkModel () const [protected], [inherited]
```

#### Returns

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

```
0.15.564.4.23 cleanModel() void smtrat::Module::cleanModel () const [inline], [protected], [inherited]
```

Substitutes variable occurrences by its model value in the model values of other variables.

```
0.15.564.4.24 clearLemmas() void smtrat::Module::clearLemmas () [inline], [inherited]
```

Deletes all yet found lemmas.

```
0.15.564.4.25 clearModel() void smtrat::Module::clearModel () const [inline], [protected], [inherited]
```

Clears the assignment, if any was found.

```
0.15.564.4.26 clearModels() void smtrat::Module::clearModels () const [inline], [protected], [inherited]
```

Clears all assignments, if any was found.

```
0.15.564.4.27 clearPassedFormula() void smtrat::Module::clearPassedFormula () [protected], [inherited]
```

```
0.15.564.4.28 collectOrigins() [1/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

---

**0.15.564.4.29 collectOrigins() [2/2]** void smtrat::Module::collectOrigins ( const FormulaT & \_formula, FormulasT & \_origins ) const [inherited]

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                 |                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------|
| <i>_formula</i> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <i>_origins</i> | The set in which to store the origins.                                                               |

**0.15.564.4.30 collectTheoryPropagations()** void smtrat::Module::collectTheoryPropagations ( ) [inherited]

**0.15.564.4.31 constraintsToInform()** const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform ( ) const [inline], [inherited]

#### Returns

The constraints which the backends must be informed about.

**0.15.564.4.32 currentlySatisfied()** virtual unsigned smtrat::Module::currentlySatisfied ( const FormulaT & ) const [inline], [virtual], [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings >](#)

**0.15.564.4.33 currentlySatisfiedByBackend()** unsigned smtrat::Module::currentlySatisfiedByBackend ( const FormulaT & \_formula ) const [inherited]

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.564.4.34 deinform()** void smtrat::Module::deinform ( const FormulaT & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                    |                                                         |
|--------------------|---------------------------------------------------------|
| <i>_constraint</i> | The constraint to remove from internal data structures. |
|--------------------|---------------------------------------------------------|

**0.15.564.4.35 `deinformCore()`** `virtual void smtrat::Module::deinformCore (`  
 `const FormulaT & _constraint )` [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

#### Parameters

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings4 >](#).

**0.15.564.4.36 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (`  
 `const std::vector< FormulaT > & origins )` const [protected], [inherited]

#### Parameters

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

#### Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.564.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (`  
 `ModuleInput::iterator _subformula,`  
 `bool _ignoreOrigins = false )` [protected], [virtual], [inherited]

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.  
Afterwards the sub-formula is removed from the passed formula.

#### Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

#### Returns

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.564.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( )` const [inherited]

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.564.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (`  
 `const std::vector< FormulaT > & _origins )` const [protected], [inherited]

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

**0.15.564.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]`

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.564.4.41 `firstUncheckedReceivedSubformula()`** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.564.4.42 `freeSplittingVariable()`** `static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable) [inline], [static], [inherited]`

**0.15.564.4.43 `generateTrivialInfeasibleSubset()`** `void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.564.4.44 `getBackendsAllModels()`** `void smtrat::Module::getBackendsAllModels () const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.564.4.45 `getBackendsModel()`** `void smtrat::Module::getBackendsModel () const [protected], [inherited]`

**0.15.564.4.46 `getInfeasibleSubsets() [1/2]`** `void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.564.4.47 `getInfeasibleSubsets() [2/2]`** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (const Module & _backend) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.564.4.48 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.564.4.49 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.564.4.50 `getOrigins()` [1/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.564.4.51 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.564.4.52 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.564.4.53 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT >` `smtrat::Module::getReceivedFormulaSimplified( )` [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.564.4.54 `hasLemmas()`** `bool smtrat::Module::hasLemmas( )` [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.564.4.55 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const` [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.564.4.56 `id()`** `std::size_t smtrat::Module::id( ) const` [inline], [inherited]

**Returns**

A unique ID to identify this module instance.

**0.15.564.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const` [inline], [inherited]

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.564.4.58 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint )` [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before `assertSubformula` is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.564.4.59 informBackends()** void smrat::Module::informBackends ( const FormulaT & \_constraint ) [inline], [protected], [inherited]

Informs all backends of this module about the given constraint.

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**0.15.564.4.60 informCore()** virtual bool smrat::Module::informCore (

const FormulaT & \_constraint ) [inline], [protected], [virtual], [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.564.4.61 informedConstraints()** const carl::FastSet<FormulaT>& smrat::Module::informedConstraints ( ) const [inline], [inherited]

**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.564.4.62 init()** void smrat::Module::init ( ) [virtual], [inherited]

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

**0.15.564.4.63 is\_minimizing()** bool smrat::Module::is\_minimizing ( ) const [inline], [inherited]

**0.15.564.4.64 `isLemmaLevel()`** `bool smtrat::Module::isLemmaLevel (``LemmaLevel level ) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

**Parameters**

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.564.4.65 `isPreprocessor()`** `bool smtrat::Module::isPreprocessor ( ) const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.564.4.66 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas ( ) const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.564.4.67 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (``const std::vector< FormulaT > & _vecSetA,``const std::vector< FormulaT > & _vecSetB ) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                       |                                  |
|-----------------------|----------------------------------|
| <code>_vecSetA</code> | A vector of sets of constraints. |
| <code>_vecSetB</code> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.564.4.68 `model()`** `const Model& smtrat::Module::model ( ) const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.564.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (``const Model & _modelA,``const Model & _modelB ) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                      |                                |
|----------------------|--------------------------------|
| <code>_modelA</code> | The first model to check for.  |
| <code>_modelB</code> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.564.4.70 `moduleName()`** `template<typename Settings >`  
`std::string smtrat::UFCegarModule< Settings >::moduleName ( ) const [inline], [virtual]`

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.564.4.71 `objective()`** `carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]`

**0.15.564.4.72 `originInReceivedFormula()`** `bool smtrat::Module::originInReceivedFormula (`  
`const FormulaT & _origin ) const [protected], [inherited]`

**0.15.564.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smtrat::Module::passedFormulaBegin (`  
`) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.564.4.74 `passedFormulaEnd()`** `ModuleInput::iterator smtrat::Module::passedFormulaEnd ( )`  
`[inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.564.4.75 `pPassedFormula()`** `const ModuleInput* smtrat::Module::pPassedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.564.4.76 `pReceivedFormula()`** `const ModuleInput* smtrat::Module::pReceivedFormula ( ) const`  
`[inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.564.4.77 `print()`** `void smtrat::Module::print (`  
`const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.564.4.78 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.564.4.79 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.564.4.80 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.564.4.81 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.564.4.82 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.564.4.83 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.564.4.84 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline], [inherited]
```

Notifies that the received formulas has been checked.

```
0.15.564.4.85 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs<= InfeasibleSubset (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.564.4.86 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.564.4.87 remove() void smtrat::Module::remove (
```

```
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

```
0.15.564.4.88 removeCore() template<typename Settings >
void smtrat::UFCegarModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.564.4.89 `removeOrigin()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.564.4.90 `removeOrigins()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (`

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.564.4.91 `rPassedFormula()`** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const`  
`[inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.564.4.92 `rReceivedFormula()`** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const`  
`[inline], [inherited]`

#### Returns

A reference to the formula passed to this module.

**0.15.564.4.93 `runBackends() [1/2]`** `virtual Answer smtrat::Module::runBackends ( ) [inline],`  
`[protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.564.4.94 `runBackends() [2/2]`** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.564.4.95 setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]  
Sets this module's unique ID to identify itself.

#### Parameters

|                                                      |                                    |
|------------------------------------------------------|------------------------------------|
| $\leftarrow$<br>$\overline{\leftarrow}$<br><i>id</i> | The id to set this module's id to. |
|------------------------------------------------------|------------------------------------|

**0.15.564.4.96 setThreadPriority()** void smtrat::Module::setThreadPriority ( thread\_priority \_threadPriority ) [inline], [inherited]  
Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                        |                                                       |
|------------------------|-------------------------------------------------------|
| <i>_threadPriority</i> | The priority to set this module's thread priority to. |
|------------------------|-------------------------------------------------------|

**0.15.564.4.97 solverState()** Answer smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.564.4.98 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]  
Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                           |                                                  |
|---------------------------|--------------------------------------------------|
| <i>_unequalConstraint</i> | A constraint having the relation symbol $\neq$ . |
|---------------------------|--------------------------------------------------|

**0.15.564.4.99 threadPriority()** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.564.4.100 updateAllModels()** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]  
Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.564.4.101 updateLemmas()** void smtrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.564.4.102 updateModel()** template<typename Settings >  
void smtrat::UFCegarModule< Settings >::updateModel () const [virtual]  
Updates the current assignment into the model.  
Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.564.4.103 usedBackends()** const std::vector<[Module](#)\*>& smtrat::Module::usedBackends ()  
const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.564.5 Field Documentation

**0.15.564.5.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected],  
[inherited]

The backends of this module which have been used.

**0.15.564.5.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected],  
[inherited]

Stores all satisfying assignments.

**0.15.564.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer  
[protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.564.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform  
[protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.564.5.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.564.5.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.564.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smtrat::Module::mFirstSubformulaToPass  
[protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.564.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`  
 Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.564.5.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`  
 Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.564.5.10 mFullCheck** `bool smtrat::Module::mFullCheck [protected], [inherited]`  
 false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.564.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`  
 Stores the infeasible subsets.

**0.15.564.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`  
 Stores the position of the first constraint of which no backend has been informed about.

**0.15.564.5.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`  
 Stores the last branches in a cycle buffer.

**0.15.564.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`  
 Stores the lemmas being valid formulas this module or its backends made.

**0.15.564.5.15 mModel** `Model smtrat::Module::mModel [mutable], [protected], [inherited]`  
 Stores the assignment of the current satisfiable result, if existent.

**0.15.564.5.16 mModelComputed** `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`  
 True, if the model has already been computed.

**0.15.564.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`  
 The number of different variables to consider for a probable infinite loop of branchings.

**0.15.564.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
 Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.564.5.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables [static], [inherited]`  
 Reusable splitting variables.

**0.15.564.5.20 mpManager** `Manager* const smtrat::Module::mpManager [protected], [inherited]`  
A reference to the manager.

**0.15.564.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.564.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.564.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheory← Propagations [protected], [inherited]`

**0.15.564.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.564.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`  
Maps variables to the number of their occurrences.

## 0.15.565 smtrat::UFRewriter Struct Reference

```
#include <EQPreprocessingUFRewriter.h>
```

### Public Types

- `typedef carl::UVariable UVariable`
- `typedef carl::UFInstance UFInstance`
- `typedef carl::UninterpretedFunction UFunction`
- `typedef carl::UTerm UTerm`
- `typedef carl::UEquality UEquality`

### Public Member Functions

- `FormulaT rewrite_ueq (const FormulaT &formula)`
- `const FormulaSetT & congruences () const noexcept`
- `const std::unordered_map< UFInstance, UVariable > & UFItoVar () const noexcept`

### 0.15.565.1 Member Typedef Documentation

**0.15.565.1.1 UEquality** `typedef carl::UEquality smtrat::UFRewriter::UEquality`

**0.15.565.1.2 UFInstance** `typedef carl::UFInstance smtrat::UFRewriter::UFInstance`

**0.15.565.1.3 UFunction** `typedef carl::UninterpretedFunction smtrat::UFRewriter::UFunction`

**0.15.565.1.4 UTerm** `typedef carl::UTerm smtrat::UFRewriter::UTerm`

**0.15.565.1.5 UVariable** `typedef carl::UVariable smtrat::UFRewriter::UVariable`

## 0.15.565.2 Member Function Documentation

**0.15.565.2.1 congruences()** `const FormulaSetT& smtrat::UFRewriter::congruences () const [inline], [noexcept]`

**0.15.565.2.2 rewrite\_ueq()** `FormulaT smtrat::UFRewriter::rewrite_ueq (const FormulaT & formula) [inline]`

**0.15.565.2.3 UFItoVar()** `const std::unordered_map<UFIInstance, UVariable>& smtrat::UFRewriter::leftrightarrow_UFItoVar () const [inline], [noexcept]`

## 0.15.566 smtrat::expression::UnaryExpression Struct Reference

```
#include <ExpressionContent.h>
```

### Public Member Functions

- `UnaryExpression (UnaryType _type, Expression &&_expression)`

### Data Fields

- `UnaryType type`
- `Expression expression`

### 0.15.566.1 Constructor & Destructor Documentation

**0.15.566.1.1 UnaryExpression()** `smtrat::expression::UnaryExpression::UnaryExpression (UnaryType _type, Expression && _expression) [inline]`

### 0.15.566.2 Field Documentation

**0.15.566.2.1 expression** `Expression smtrat::expression::UnaryExpression::expression`

**0.15.566.2.2 type** `UnaryType smtrat::expression::UnaryExpression::type`

## 0.15.567 smtrat::cad::debug::UnifiedData Struct Reference

```
#include <TikzHistoryPrinter.h>
```

## Public Member Functions

- void `add` (std::size\_t step, const std::string &`data`)
- bool `showsOn` (std::size\_t step) const
- const std::string & `data` (std::size\_t step) const

## Data Fields

- std::vector< std::optional< std::string > > `steps`

### 0.15.567.1 Member Function Documentation

**0.15.567.1.1 `add()`** void smtrat::cad::debug::UnifiedData::add ( std::size\_t *step*, const std::string & *data* ) [inline]

**0.15.567.1.2 `data()`** const std::string& smtrat::cad::debug::UnifiedData::data ( std::size\_t *step* ) const [inline]

**0.15.567.1.3 `showsOn()`** bool smtrat::cad::debug::UnifiedData::showsOn ( std::size\_t *step* ) const [inline]

### 0.15.567.2 Field Documentation

**0.15.567.2.1 `steps`** std::vector<std::optional<std::string>> smtrat::cad::debug::UnifiedData::  
:steps

## 0.15.568 smtrat::parser::types::UninterpretedTheory Struct Reference

Types of the theory of equalities and uninterpreted functions.

```
#include <TheoryTypes.h>
```

### Public Types

- typedef mpl::vector< carl::UTerm > `ConstTypes`
- typedef mpl::vector< carl::UVariable > `VariableTypes`
- typedef mpl::vector< carl::UTerm > `ExpressionTypes`
- typedef mpl::vector< carl::UTerm, carl::UVariable > `TermTypes`
- typedef carl::mpl\_variant\_of< `TermTypes` >::type `TermType`

### 0.15.568.1 Detailed Description

Types of the theory of equalities and uninterpreted functions.

### 0.15.568.2 Member Typedef Documentation

**0.15.568.2.1 `ConstTypes`** typedef mpl::vector<carl::UTerm> smtrat::parser::types::UninterpretedTheory::ConstType

**0.15.568.2.2 ExpressionTypes** `typedef mpl::vector<carl::UTerm> smtrat::parser::types::UninterpretedTheory::Exp`

**0.15.568.2.3 TermType** `typedef carl::mpl_variant_of<TermTypes>::type smtrat::parser::types::UninterpretedTheo`

**0.15.568.2.4 TermTypes** `typedef mpl::vector<carl::UTerm, carl::UVariable> smtrat::parser::types::Uninterprete`

**0.15.568.2.5 VariableTypes** `typedef mpl::vector<carl::UVariable> smtrat::parser::types::UninterpretedTheory::Va`

## 0.15.569 smtrat::parser::UninterpretedTheory Struct Reference

Implements the theory of equalities and uninterpreted functions.

```
#include <Uninterpreted.h>
```

### Public Member Functions

- `UninterpretedTheory (ParserState *state)`
- `FormulaT coupleBooleanVariables (carl::Variable v, carl::UVariable uv)`
- `bool declareVariable (const std::string &name, const carl::Sort &sort, types::VariableType &result, TheoryError &errors)`

*Declare a new variable with the given name and the given sort.*
- `bool handleITE (const FormulaT &ifterm, const types::TermType &thenterm, const types::TermType &elseterm, types::TermType &result, TheoryError &errors)`

*Resolve an if-then-else operator.*
- `bool handleFunctionInstantiation (const carl::UninterpretedFunction &f, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)`
- `bool handleDistinct (const std::vector< types::TermType > &, types::TermType &, TheoryError &errors)`

*Resolve a distinct operator.*
- `bool functionCall (const Identifier &identifier, const std::vector< types::TermType > &arguments, types::TermType &result, TheoryError &errors)`

*Resolve another unknown function call.*
- `virtual bool resolveSymbol (const Identifier &, types::TermType &, TheoryError &errors)`

*Resolve a symbol that was not declared within the ParserState.*
- `template<typename T, typename Builder> FormulaT expandDistinct (const std::vector< T > &values, const Builder &neqBuilder)`
- `virtual bool instantiate (const types::VariableType &, const types::TermType &, types::TermType &, TheoryError &errors)`

*Instantiate a variable within a term.*
- `virtual bool refreshVariable (const types::VariableType &, types::VariableType &, TheoryError &errors)`

### Static Public Member Functions

- `static void addSimpleSorts (qi::symbols< char, carl::Sort > &)`

*Initialize the global symbol table for simple sorts.*
- `static void addConstants (qi::symbols< char, types::ConstType > &)`

*Initialize the global symbol table for constants.*

### Data Fields

- `std::map< types::TermType, carl::UTerm > mInstantiatedArguments`
- `carl::Sort mBoolSort`
- `carl::UVariable mTrue`
- `carl::UVariable mFalse`
- `ParserState * state`

### 0.15.569.1 Detailed Description

Implements the theory of equalities and uninterpreted functions.

### 0.15.569.2 Constructor & Destructor Documentation

**0.15.569.2.1 UninterpretedTheory()** smtrat::parser::UninterpretedTheory::UninterpretedTheory (   
   ParserState \* state )

### 0.15.569.3 Member Function Documentation

**0.15.569.3.1 addConstants()** static void smtrat::parser::AbstractTheory::addConstants (   
   qi::symbols< char, types::ConstType > & ) [inline], [static], [inherited]  
Initialize the global symbol table for constants.

**0.15.569.3.2 addSimpleSorts()** static void smtrat::parser::AbstractTheory::addSimpleSorts (   
   qi::symbols< char, carl::Sort > & ) [inline], [static], [inherited]  
Initialize the global symbol table for simple sorts.

**0.15.569.3.3 coupleBooleanVariables()** FormulaT smtrat::parser::UninterpretedTheory::coupleBooleanVariables (   
   carl::Variable v,   
   carl::UVariable uv ) [inline]

**0.15.569.3.4 declareVariable()** bool smtrat::parser::UninterpretedTheory::declareVariable (   
   const std::string &,   
   const carl::Sort &,   
   types::VariableType &,   
   TheoryError & errors ) [virtual]

Declare a new variable with the given name and the given sort.

Reimplemented from [smtrat::parser::AbstractTheory](#).

**0.15.569.3.5 expandDistinct()** template<typename T, typename Builder>   
 FormulaT smtrat::parser::AbstractTheory::expandDistinct (   
   const std::vector< T > & values,   
   const Builder & negBuilder ) [inline], [inherited]

**0.15.569.3.6 functionCall()** bool smtrat::parser::UninterpretedTheory::functionCall (   
   const Identifier &,   
   const std::vector< types::TermType > &,   
   types::TermType &,   
   TheoryError & errors ) [virtual]

Resolve another unknown function call.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.569.3.7 handleDistinct() bool smtrat::parser::UninterpretedTheory::handleDistinct (
 const std::vector< types::TermType > & ,
 types::TermType & ,
 TheoryError & errors) [virtual]
```

Resolve a distinct operator.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.569.3.8 handleFunctionInstantiation() bool smtrat::parser::UninterpretedTheory::handleFunctionInstantiation (
 const carl::UninterpretedFunction & f,
 const std::vector< types::TermType > & arguments,
 types::TermType & result,
 TheoryError & errors)
```

```
0.15.569.3.9 handleITE() bool smtrat::parser::UninterpretedTheory::handleITE (
 const FormulaT & ,
 const types::TermType & ,
 const types::TermType & ,
 types::TermType & ,
 TheoryError & errors) [virtual]
```

Resolve an if-then-else operator.

Reimplemented from [smtrat::parser::AbstractTheory](#).

```
0.15.569.3.10 instantiate() virtual bool smtrat::parser::AbstractTheory::instantiate (
 const types::VariableType & ,
 const types::TermType & ,
 types::TermType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Instantiate a variable within a term.

Reimplemented in [smtrat::parser::BitvectorTheory](#), and [smtrat::parser::ArithmeticTheory](#).

```
0.15.569.3.11 refreshVariable() virtual bool smtrat::parser::AbstractTheory::refreshVariable (
 const types::VariableType & ,
 types::VariableType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Reimplemented in [smtrat::parser::BitvectorTheory](#).

```
0.15.569.3.12 resolveSymbol() virtual bool smtrat::parser::AbstractTheory::resolveSymbol (
 const Identifier & ,
 types::TermType & ,
 TheoryError & errors) [inline], [virtual], [inherited]
```

Resolve a symbol that was not declared within the [ParserState](#).

This might be a symbol that actually uses indices, for example bitvector constants.

Reimplemented in [smtrat::parser::BitvectorTheory](#).

## 0.15.569.4 Field Documentation

**0.15.569.4.1 mBoolSort** carl::Sort smtrat::parser::UninterpretedTheory::mBoolSort

**0.15.569.4.2 mFalse** carl::UVariable smtrat::parser::UninterpretedTheory::mFalse

**0.15.569.4.3 mInstantiatedArguments** std::map<types::TermType, carl::UTerm> smtrat::parser::UninterpretedTheory::mInstantiatedArguments

**0.15.569.4.4 mTrue** carl::UVariable smtrat::parser::UninterpretedTheory::mTrue

**0.15.569.4.5 state** ParserState\* smtrat::parser::AbstractTheory::state [inherited]

## 0.15.570 smtrat::union\_find< T, no\_compression, MaxTracebackLength > Class Template Reference

Implements a union-find datastructure, a datastructure containing an array of entries.

```
#include <union_find.h>
```

### Public Member Functions

- **union\_find ()**  
*Construct union-find datastructure with an empty array of entries.*
- **union\_find (std::size\_t size\_)**  
*Construct union-find datastructure with an array of size\_ default-constructed elements.*
- template<typename InputIterator>  
**union\_find (InputIterator begin\_, typename std::enable\_if<!std::is\_void< T >::value, InputIterator>::type end\_)**  
*Construct union-find datastructure, copying the sequence identified by [begin\_,end\_) into the array.*
- **union\_find (const union\_find &other)**
- **union\_find & operator= (const union\_find &other)**
- **std::size\_t find (std::size\_t index\_)** const noexcept  
*Find the index of the representative of the entry at index index\_.*
- **bool equivalent (std::size\_t index1, std::size\_t index2)** const noexcept  
*Determines if the two entries at positions index1 and index2 are considered equivalent, that is find(index1) == find(index2).*
- **bool union\_ (std::size\_t index1, std::size\_t index2)** noexcept  
*Make the entries identified by index1 and index2 equivalent.*
- **std::size\_t fast\_union (std::size\_t rep1, std::size\_t rep2)**  
*As union\_, but does NOT compute find(rep1), find(rep2), but assumes both indices are already pointing to representatives.*
- template<bool Ignore = true>  
**std::enable\_if<!std::is\_void< T >::value &&Ignore, std::size\_t>::type add (const T &value)**  
*Add a new default-constructed value to the end of this union find array.*
- **void set\_class (std::size\_t i, std::size\_t c)**
- **std::size\_t rank (std::size\_t i)**  
*The rank (upper bound on the height of the tree rooted at i) of the entry at index i.*
- template<bool Ignore = true>  
**std::enable\_if<!std::is\_void< T >::value &&Ignore, T>::type & operator[] (std::size\_t index\_)** noexcept  
*Access to values (only if T is not void, relies on C++11 feature: default template arguments for template methods).*
- template<bool Ignore = true>  
**std::enable\_if<!std::is\_void< T >::value &&Ignore, const T>::type & operator[] (std::size\_t index\_)** const noexcept  
*Access to values (only if T is not void, relies on C++11 feature: default template arguments for template methods), const.*

- `std::size_t size () const noexcept`  
*Returns the number of entries in this union-find datastructure.*
- `template<bool Ignore = true>`  
`std::enable_if<!std::is_void< T >::value &&Ignore, void >::type initialize (std::size_t new_size)`  
*Initialize the union-find datastructure with an array of new\_size default-constructed entries, version for T != void.*
- `template<bool Ignore = true>`  
`std::enable_if< std::is_void< T >::value &&Ignore, void >::type initialize (std::size_t new_size)`  
*Initialize the union-find datastructure with an array of new\_size default-constructed entries, version for T == void.*
- `template<typename InputIterator >`  
`std::enable_if<!std::is_void< T >::value &&!std::is_void< InputIterator >::value, void >::type initialize (InputIterator begin_, InputIterator end_)`  
*Initialize the union-find datastructure with an array of entries copied from the sequence [begin\_,end\_].*

### 0.15.570.1 Detailed Description

```
template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
class smtrat::union_find< T, no_compression, MaxTracebackLength >
```

Implements a union-find datastructure, a datastructure containing an array of entries.

Basically, this is an array of elements of type T with the following extra structure: Every entry is contained in an equivalence class identified by a single representative entry. In order to find this representative, every element has an associated parent entry; if this is the entry itself, the entry is the representative of its equivalence class. Using a rank count (basically a lower bound for the height of the tree below an entry), the height of the trees below the representative entries is ALWAYS less than  $\text{ceil}(\log_2(\#\text{entries}))$ . Moreover, while finding the parent of an entry, all entries on the path to the root are set to directly point to the root element. This improves the amortized number of steps used to find the representative of an entry from  $O(\log_2(n))$  to  $O(a(n))$  where  $a(n)$  is the inverse of the Ackermann function. All this extra structure is embedded directly within the array.

### 0.15.570.2 Constructor & Destructor Documentation

**0.15.570.2.1 union\_find() [1/4]** `template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>`  
`smtrat::union_find< T, no_compression, MaxTracebackLength >::union_find ( ) [inline]`  
Construct union-find datastructure with an empty array of entries.

**0.15.570.2.2 union\_find() [2/4]** `template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>`  
`smtrat::union_find< T, no_compression, MaxTracebackLength >::union_find (`  
`std::size_t size_ ) [inline], [explicit]`  
Construct union-find datastructure with an array of size\_ default-constructed elements.

**0.15.570.2.3 union\_find() [3/4]** `template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>`  
`template<typename InputIterator >`  
`smtrat::union_find< T, no_compression, MaxTracebackLength >::union_find (`  
`InputIterator begin_,`  
`typename std::enable_if< !std::is_void< T >::value, InputIterator >::type end_ )`  
`[inline]`  
Construct union-find datastructure, copying the sequence identified by [begin\_,end\_] into the array.

```
0.15.570.2.4 union_find() [4/4] template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
smtrat::union_find< T, no_compression, MaxTracebackLength >::union_find (
 const union_find< T, no_compression, MaxTracebackLength > & other) [inline]
```

### 0.15.570.3 Member Function Documentation

```
0.15.570.3.1 add() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
template<bool Ignore = true>
std::enable_if< !std::is_void<T>::value && Ignore, std::size_t>::type smtrat::union_find< T,
no_compression, MaxTracebackLength >::add (
 const T & value) [inline]
```

Add a new default-constructed value to the end of this union find array.

This does not check for the uniqueness of this entry. This method can not be used if T is void. Returns the index associated with the new entry (i.e. the size of the UF structure before the call).

```
0.15.570.3.2 equivalent() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
bool smtrat::union_find< T, no_compression, MaxTracebackLength >::equivalent (
 std::size_t index1,
 std::size_t index2) const [inline], [noexcept]
```

Determines if the two entries at positions index1 and index2 are considered equivalent, that is find(index1) == find(index2).

```
0.15.570.3.3 fast_union() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
std::size_t smtrat::union_find< T, no_compression, MaxTracebackLength >::fast_union (
 std::size_t rep1,
 std::size_t rep2) [inline]
```

As union\_, but does NOT compute find(rep1), find(rep2), but assumes both indices are already pointing to representatives.

Returns the new representative of both elements after the operation; this will either be rep1 or rep2.

```
0.15.570.3.4 find() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
std::size_t smtrat::union_find< T, no_compression, MaxTracebackLength >::find (
 std::size_t index_) const [inline], [noexcept]
```

Find the index of the representative of the entry at index index\_.

Note that this method is const-qualified as it does not change the observable behaviour of this class, but it is not bitwise-const due to the path compression behaviour. Therefore it is not safe to call this method from different threads on the same UF datastructure without external synchronization.

```
0.15.570.3.5 initialize() [1/3] template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
template<typename InputIterator >
std::enable_if<!std::is_void<T>::value && !std::is_void<InputIterator>::value, void>::type
smtrat::union_find< T, no_compression, MaxTracebackLength >::initialize (
 InputIterator begin_,
 InputIterator end_) [inline]
```

Initialize the union-find datastructure with an array of entries copied from the sequence [begin\_,end\_].

```
0.15.570.3.6 initialize() [2/3] template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
template<bool Ignore = true>
std::enable_if<!std::is_void<T>::value && Ignore, void>::type smtrat::union_find< T, no_compression, MaxTracebackLength >::initialize (
 std::size_t new_size) [inline]
```

Initialize the union-find datastructure with an array of new\_size default-constructed entries, version for T != void.

```
0.15.570.3.7 initialize() [3/3] template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
template<bool Ignore = true>
std::enable_if<std::is_void<T>::value && Ignore, void>::type smtrat::union_find< T, no_compression, MaxTracebackLength >::initialize (
 std::size_t new_size) [inline]
```

Initialize the union-find datastructure with an array of new\_size default-constructed entries, version for T == void.

```
0.15.570.3.8 operator=() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
union_find& smtrat::union_find< T, no_compression, MaxTracebackLength >::operator= (
 const union_find< T, no_compression, MaxTracebackLength > & other) [inline]
```

```
0.15.570.3.9 operator[]() [1/2] template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
template<bool Ignore = true>
std::enable_if< !std::is_void<T>::value && Ignore, const T>::type& smtrat::union_find< T, no_compression, MaxTracebackLength >::operator[] (
 std::size_t index_) const [inline], [noexcept]
```

Access to values (only if T is not void, relies on C++11 feature: default template arguments for template methods), const.

```
0.15.570.3.10 operator[]() [2/2] template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
template<bool Ignore = true>
std::enable_if< !std::is_void<T>::value && Ignore, T>::type& smtrat::union_find< T, no_compression, MaxTracebackLength >::operator[] (
 std::size_t index_) [inline], [noexcept]
```

Access to values (only if T is not void, relies on C++11 feature: default template arguments for template methods).

```
0.15.570.3.11 rank() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
std::size_t smtrat::union_find< T, no_compression, MaxTracebackLength >::rank (
 std::size_t i) [inline]
```

The rank (upper bound on the height of the tree rooted at i) of the entry at index i.

```
0.15.570.3.12 set_class() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
void smtrat::union_find< T, no_compression, MaxTracebackLength >::set_class (
 std::size_t i,
 std::size_t c) [inline]
```

```
0.15.570.3.13 size() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
std::size_t smtrat::union_find< T, no_compression, MaxTracebackLength >::size () const [inline], [noexcept]
```

Returns the number of entries in this union-find datastructure.

```
0.15.570.3.14 union_() template<typename T = void, bool no_compression = true, std::size_t MaxTracebackLength = 64>
bool smtrat::union_find< T, no_compression, MaxTracebackLength >::union_ (
 std::size_t index1,
 std::size_t index2) [inline], [noexcept]
```

Make the entries identified by index1 and index2 equivalent.

As this is determined by the union-by-rank mechanism, either one of find(index1) or find(index2) can become the index of the new representative of both entries. Returns false when this operation did no changes on the datastructure and true otherwise.

## 0.15.571 smtrat::UnionFindInterface< T, Implementation > Struct Template Reference

```
#include <UnionFind.h>
```

### Public Types

- using **Value** = typename Implementation::Value
- using **Representative** = typename Implementation::Representative
- using **TranslateMap** = std::unordered\_map< T, **Value** >

### Public Member Functions

- **UnionFindInterface** (**TranslateMap** &translate)
- **Value insert\_value** (T const &var) noexcept
- template<typename Container >
void **init** (Container const &cont) noexcept
- bool **has\_variable** (T const &var) noexcept
- auto **find** (T const &val) noexcept -> **Representative**
- void **merge** (T const &a, T const &b) noexcept
- void **backtrack** (T const &a, T const &b) noexcept

### 0.15.571.1 Member Typedef Documentation

```
0.15.571.1.1 Representative template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::Representative = typename Implementation::Representative
```

```
0.15.571.1.2 TranslateMap template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::TranslateMap = std::unordered_map<T, Value>
```

```
0.15.571.1.3 Value template<typename T , typename Implementation >
using smtrat::UnionFindInterface< T, Implementation >::Value = typename Implementation::Value
```

### 0.15.571.2 Constructor & Destructor Documentation

```
0.15.571.2.1 UnionFindInterface() template<typename T , typename Implementation >
smtrat::UnionFindInterface< T, Implementation >::UnionFindInterface (
 TranslateMap & translate) [inline]
```

### 0.15.571.3 Member Function Documentation

```
0.15.571.3.1 backtrack() template<typename T , typename Implementation >
void smtrat::UnionFindInterface< T, Implementation >::backtrack (
 T const & a,
 T const & b) [inline], [noexcept]
```

```
0.15.571.3.2 find() template<typename T , typename Implementation >
auto smtrat::UnionFindInterface< T, Implementation >::find (
 T const & val) -> Representative [inline], [noexcept]
```

```
0.15.571.3.3 has_variable() template<typename T , typename Implementation >
bool smtrat::UnionFindInterface< T, Implementation >::has_variable (
 T const & var) [inline], [noexcept]
```

```
0.15.571.3.4 init() template<typename T , typename Implementation >
template<typename Container >
void smtrat::UnionFindInterface< T, Implementation >::init (
 Container const & cont) [inline], [noexcept]
```

```
0.15.571.3.5 insert_value() template<typename T , typename Implementation >
Value smtrat::UnionFindInterface< T, Implementation >::insert_value (
 T const & var) [inline], [noexcept]
```

```
0.15.571.3.6 merge() template<typename T , typename Implementation >
void smtrat::UnionFindInterface< T, Implementation >::merge (
 T const & a,
 T const & b) [inline], [noexcept]
```

## 0.15.572 smtrat::UnionFindModule< Settings > Class Template Reference

```
#include <UnionFindModule.h>
```

### Public Types

- **typedef Settings SettingsType**
- **enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }**

### Public Member Functions

- **std::string moduleName () const**
- **UnionFindModule (const ModuleInput \*\_formula, Conditionals &\_conditionals, Manager \*\_manager=nullptr)**
- **~UnionFindModule ()**
- **bool informCore (const FormulaT &\_constraint)**  
*Informs the module about the given constraint.*
- **bool addCore (ModuleInput::const\_iterator \_subformula)**

- The module has to take the given sub-formula of the received formula into account.*
- void **removeCore** (*ModuleInput::const\_iterator \_subformula*)  
*Removes the subformula of the received formula at the given position to the considered ones of this module.*
  - void **updateModel** () const  
*Updates the current assignment into the model.*
  - **Answer checkCore** ()  
*Checks the received formula for consistency.*
  - bool **inform** (*const FormulaT &\_constraint*)  
*Informs the module about the given constraint.*
  - void **deinform** (*const FormulaT &\_constraint*)  
*The inverse of informing about a constraint.*
  - virtual void **init** ()  
*Informs all backends about the so far encountered constraints, which have not yet been communicated.*
  - bool **add** (*ModuleInput::const\_iterator \_subformula*)  
*The module has to take the given sub-formula of the received formula into account.*
  - virtual **Answer check** (*bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_← VARIABLE*)  
*Checks the received formula for consistency.*
  - virtual void **remove** (*ModuleInput::const\_iterator \_subformula*)  
*Removes everything related to the given sub-formula of the received formula.*
  - virtual void **updateAllModels** ()  
*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*
  - virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const  
*Partition the variables from the current model into equivalence classes according to their assigned value.*
  - unsigned **currentlySatisfiedByBackend** (*const FormulaT &\_formula*) const
  - virtual unsigned **currentlySatisfied** (*const FormulaT &*) const
  - bool **receivedVariable** (*carl::Variable::Arg \_var*) const
  - **Answer solverState** () const
  - std::size\_t **id** () const
  - void **setId** (*std::size\_t \_id*)  
*Sets this modules unique ID to identify itself.*
  - **thread\_priority threadPriority** () const
  - void **setThreadPriority** (*thread\_priority \_threadPriority*)  
*Sets the priority of this module to get a thread for running its check procedure.*
  - const **ModuleInput \* pReceivedFormula** () const
  - const **ModuleInput & rReceivedFormula** () const
  - const **ModuleInput \* pPassedFormula** () const
  - const **ModuleInput & rPassedFormula** () const
  - const **Model & model** () const
  - const std::vector< **Model** > & **allModels** () const
  - const std::vector< **FormulaSetT** > & **infeasibleSubsets** () const
  - const std::vector< **Module** \* > & **usedBackends** () const
  - const carl::FastSet< **FormulaT** > & **constraintsToInform** () const
  - const carl::FastSet< **FormulaT** > & **informedConstraints** () const
  - void **addLemma** (*const FormulaT &\_lemma, const LemmaType &\_lt=LemmaType::NORMAL, const FormulaT &\_preferredFormula=FormulaT(carl::FormulaType::TRUE)*)  
*Stores a lemma being a valid formula.*
  - bool **hasLemmas** ()  
*Checks whether this module or any of its backends provides any lemmas.*
  - void **clearLemmas** ()  
*Deletes all yet found lemmas.*
  - const std::vector< **Lemma** > & **lemmas** () const

- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- `void receivedFormulaChecked ()`  
*Notifies that the received formulas has been checked.*
- `const smrat::Conditionals & answerFound () const`
- `bool isPreprocessor () const`
- `carl::Variable objective () const`
- `bool is_minimizing () const`
- `void excludeNotReceivedVariablesFromModel () const`  
*Excludes all variables from the current model, which do not occur in the received formula.*
- `void updateLemmas ()`  
*Stores all lemmas of any backend of this module in its own lemma vector.*
- `void collectTheoryPropagations ()`
- `void collectOrigins (const FormulaT &_formula, FormulasT &_origins) const`  
*Collects the formulas in the given formula, which are part of the received formula.*
- `void collectOrigins (const FormulaT &_formula, FormulaSetT &_origins) const`
- `bool hasValidInfeasibleSubset () const`
- `void checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infssubset, const std::string &_filename="smaller_muses", unsigned _maxSizeDifference=1) const`  
*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- `virtual std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- `void print (const std::string &_initiation="***") const`  
*Prints everything relevant of the solver.*
- `void printReceivedFormula (const std::string &_initiation="***") const`  
*Prints the vector of the received formula.*
- `void printPassedFormula (const std::string &_initiation="***") const`  
*Prints the vector of passed formula.*
- `void printInfeasibleSubsets (const std::string &_initiation="***") const`  
*Prints the infeasible subsets.*
- `void printModel (std::ostream &_out=std::cout) const`  
*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- `void printAllModels (std::ostream &_out=std::cout)`  
*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- `static void freeSplittingVariable (const FormulaT &_splittingVariable)`

## Static Public Attributes

- `static size_t mNumOfBranchVarsToStore = 5`  
*The number of different variables to consider for a probable infinite loop of branchings.*
- `static std::vector< Branching > mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))`  
*Stores the last branches in a cycle buffer.*
- `static size_t mFirstPosInLastBranches = 0`  
*The beginning of the cyclic buffer storing the last branches.*
- `static std::vector< FormulaT > mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual void `deinformCore` (const `FormulaT` &\_constraint)
 

*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const
 

*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const
 

*Clears the assignment, if any was found.*
- void `clearModels` () const
 

*Clears all assignments, if any was found.*
- void `cleanModel` () const
 

*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin` ()
- `ModuleInput::iterator passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
 

*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const `FormulaT` & `getOrigins` (`ModuleInput::const_iterator` \_formula) const
 

*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const `FormulaT` &\_formula, `FormulasT` &\_origins) const
- void `getOrigins` (const `FormulaT` &\_formula, `FormulaSetT` &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const `FormulaT` &\_origin)
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
- void `informBackends` (const `FormulaT` &\_constraint)
 

*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const `FormulaT` &\_constraint)
 

*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)
 

*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const `FormulaT` &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula)
 

*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const std::shared\_ptr< std::vector< `FormulaT` >> &\_origins)
 

*Adds the given formula to the passed formula.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const `FormulaT` &\_formula, const `FormulaT` &\_origin)
 

*Adds the given formula to the passed formula.*
- void `generateTrivialInfeasibleSubset` ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void `receivedFormulasAsInfeasibleSubset` (`ModuleInput::const_iterator` \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< `FormulaT` >::const\_iterator `findBestOrigin` (const std::vector< `FormulaT` > &\_origins) const
- void `getInfeasibleSubsets` ()
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< `FormulaSetT` > `getInfeasibleSubsets` (const `Module` &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const `Model` & `backendsModel` () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void `getBackendsModel` () const

- void `getBackendsAllModels () const`  
*Stores all models of a backend in the list of all models of this module.*
- virtual `Answer runBackends (bool _final, bool _full, carl::Variable _objective)`  
*Runs the backend solvers on the passed formula.*
- virtual `Answer runBackends ()`
- virtual `ModuleInput::iterator eraseSubformulaFromPassedFormula (ModuleInput::iterator _subformula, bool _ignoreOrigins=false)`  
*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void `clearPassedFormula ()`
- std::vector< `FormulaT` > `merge` (const std::vector< `FormulaT` > &, const std::vector< `FormulaT` > &) const  
*Merges the two vectors of sets into the first one.*
- size\_t `determine_smallest_origin` (const std::vector< `FormulaT` > &origins) const
- bool `probablyLooping` (const typename Poly::PolyType &\_branchingPolynomial, const Rational &\_branchingValue) const  
*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)  
*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool `branchAt` (const Poly &\_polynomial, bool \_integral, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &&\_premise=std::vector< `FormulaT` >())
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, std::vector< `FormulaT` > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
- bool `branchAt` (carl::Variable::Arg \_var, const Rational &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< `FormulaT` > &\_premise=std::vector< `FormulaT` >())
- void `splitUnequalConstraint` (const `FormulaT` &)  
*Adds the following lemmas for the given constraint p!=0.*
- unsigned `checkModel () const`
- void `addInformationRelevantFormula` (const `FormulaT` &formula)  
*Adds a formula to the InformationRelevantFormula.*
- const std::set< `FormulaT` > & `getInformationRelevantFormulas ()`  
*Gets all InformationRelevantFormulas.*
- bool `isLemmaLevel (LemmaLevel level)`  
*Checks if current lemma level is greater or equal to given level.*

### Static Protected Member Functions

- static bool `modelsDisjoint` (const Model &\_modelA, const Model &\_modelB)  
*Checks whether there is no variable assigned by both models.*

### Protected Attributes

- std::vector< `FormulaSetT` > `mInfeasibleSubsets`  
*Stores the infeasible subsets.*
- Manager \*const `mpManager`  
*A reference to the manager.*
- Model `mModel`  
*Stores the assignment of the current satisfiable result, if existent.*
- std::vector< Model > `mAllModels`  
*Stores all satisfying assignments.*

- bool `mModelComputed`  
*True, if the model has already been computed.*
- bool `mFinalCheck`  
*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*
- bool `mFullCheck`  
*false, if this module should avoid too expensive procedures and rather return unknown instead.*
- carl::Variable `mObjectiveVariable`  
*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*
- std::vector< `TheoryPropagation` > `mTheoryPropagations`
- std::atomic< `Answer` > `mSolverState`  
*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*
- std::atomic\_bool \* `mBackendsFoundAnswer`  
*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*
- Conditionals `mFoundAnswer`  
*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*
- std::vector< `Module` \* > `mUsedBackends`  
*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*
- std::vector< `Module` \* > `mAllBackends`  
*The backends of this module which have been used.*
- std::vector< `Lemma` > `mLemmas`  
*Stores the lemmas being valid formulas this module or its backends made.*
- ModuleInput::iterator `mFirstSubformulaToPass`  
*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*
- carl::FastSet< `FormulaT` > `mConstraintsToInform`  
*Stores the constraints which the backends must be informed about.*
- carl::FastSet< `FormulaT` > `mInformedConstraints`  
*Stores the position of the first constraint of which no backend has been informed about.*
- ModuleInput::const\_iterator `mFirstUncheckedReceivedSubformula`  
*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*
- unsigned `mSmallerMusesCheckCounter`  
*Counter used for the generation of the smt2 files to check for smaller muses.*
- std::vector< std::size\_t > `mVariableCounters`  
*Maps variables to the number of their occurrences.*

### 0.15.572.1 Member Typedef Documentation

```
0.15.572.1.1 SettingsType template<typename Settings >
typedef Settings smtrat::UnionFindModule< Settings >::SettingsType
```

### 0.15.572.2 Member Enumeration Documentation

```
0.15.572.2.1 LemmaType enum smtrat::Module::LemmaType : unsigned [strong], [inherited]
```

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.572.3 Constructor & Destructor Documentation

**0.15.572.3.1 UnionFindModule()** template<typename Settings >  
`smtrat::UnionFindModule< Settings >::UnionFindModule (`  
 `const ModuleInput * _formula,`  
 `Conditionals & _conditionals,`  
 `Manager * _manager = nullptr )`

**0.15.572.3.2 ~UnionFindModule()** template<typename Settings >  
`smtrat::UnionFindModule< Settings >::~UnionFindModule ( )`

### 0.15.572.4 Member Function Documentation

**0.15.572.4.1 add()** bool smtrat::Module::add (  
 `ModuleInput::const_iterator _subformula ) [inherited]`

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.572.4.2 addConstraintToInform()** void smtrat::Module::addConstraintToInform (  
 `const FormulaT & _constraint ) [protected], [virtual], [inherited]`

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

#### Parameters

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.572.4.3 addCore()** template<typename Settings >  
`bool smtrat::UnionFindModule< Settings >::addCore (`  
 `ModuleInput::const_iterator _subformula ) [virtual]`

The module has to take the given sub-formula of the received formula into account.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

#### Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.572.4.4 addInformationRelevantFormula()** void smtrat::Module::addInformationRelevantFormula (  
    const [FormulaT](#) & formula ) [protected], [inherited]

Adds a formula to the InformationRelevantFormula.

#### Parameters

|                |                |
|----------------|----------------|
| <i>formula</i> | Formula to add |
|----------------|----------------|

**0.15.572.4.5 addLemma()** void smtrat::Module::addLemma (  
    const [FormulaT](#) & \_lemma,  
    const [LemmaType](#) & \_lt = [LemmaType::NORMAL](#),  
    const [FormulaT](#) & \_preferredFormula = [FormulaT](#)( [carl::FormulaType::TRUE](#) ) ) [inline],  
[inherited]

Stores a lemma being a valid formula.

#### Parameters

|                          |                                                                       |
|--------------------------|-----------------------------------------------------------------------|
| <i>_lemma</i>            | The eduction/lemma to store.                                          |
| <i>_lt</i>               | The type of the lemma.                                                |
| <i>_preferredFormula</i> | Hint for the next decision, which formula should be assigned to true. |

**0.15.572.4.6 addOrigin()** void smtrat::Module::addOrigin (  
    [ModuleInput::iterator](#) \_formula,  
    const [FormulaT](#) & \_origin ) [inline], [protected], [inherited]

Adds the given set of formulas in the received formula to the origins of the given passed formula.

#### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>_formula</i> | The passed formula to set the origins for.                |
| <i>_origin</i>  | A set of formulas in the received formula of this module. |

**0.15.572.4.7 addReceivedSubformulaToPassedFormula()** std::pair<[ModuleInput::iterator](#),bool>  
smtrat::Module::addReceivedSubformulaToPassedFormula (  
    [ModuleInput::const\\_iterator](#) \_subformula ) [inline], [protected], [inherited]

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

#### Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>_subformula</i> | The sub-formula of the received formula to copy. |
|--------------------|--------------------------------------------------|

**0.15.572.4.8 addSubformulaToPassedFormula()** [1/3] std::pair<[ModuleInput::iterator](#),bool> smtrat↔

```
::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

#### Parameters

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.572.4.9 `addSubformulaToPassedFormula()` [2/3] `std::pair<ModuleInput::iterator,bool>` smrat->  
::Module::addSubformulaToPassedFormula (**

```
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.572.4.10 `addSubformulaToPassedFormula()` [3/3] `std::pair<ModuleInput::iterator,bool>`**

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.572.4.11 allModels()** const std::vector<[Model](#)>& smtrat::Module::allModels () const [inline], [inherited]

**Returns**

All satisfying assignments, if existent.

**0.15.572.4.12 anAnswerFound()** bool smtrat::Module::anAnswerFound () const [inline], [protected], [inherited]

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

**Returns**

True, if one of them has determined a result.

**0.15.572.4.13 answerFound()** const [smtrat::Conditionals](#)& smtrat::Module::answerFound () const [inline], [inherited]

**Returns**

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.572.4.14 backendsModel()** const [Model](#) & smtrat::Module::backendsModel () const [protected], [inherited]

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

**0.15.572.4.15 branchAt() [1/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const [Rational](#) & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< [FormulaT](#) > & \_premise = std::vector<[FormulaT](#)>() ) [inline], [protected], [inherited]

**0.15.572.4.16 branchAt() [2/4]** bool smtrat::Module::branchAt ( carl::Variable::Arg \_var, const [Rational](#) & \_value, std::vector< [FormulaT](#) > && \_premise, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false ) [inline], [protected], [inherited]

**0.15.572.4.17 branchAt() [3/4]** bool smtrat::Module::branchAt ( const [Poly](#) & \_polynomial, bool \_integral, const [Rational](#) & \_value, bool \_leftCaseWeak = true, bool \_preferLeftCase = true, bool \_useReceivedFormulaAsPremise = false, const std::vector< [FormulaT](#) > & \_premise = std::vector<[FormulaT](#)>() ) [inline], [protected], [inherited]

**0.15.572.4.18 branchAt() [4/4]** `bool smtrat::Module::branchAt (`

```
const Poly & _polynomial,
bool _integral,
const Rational & _value,
std::vector< FormulaT > && _premise,
bool _leftCaseWeak = true,
bool _preferLeftCase = true,
bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

**Parameters**

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

**0.15.572.4.19 check() [Answer](#)** `smtrat::Module::check (`

```
bool _final = false,
bool _full = true,
carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

**Parameters**

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.572.4.20 checkCore()** `template<typename Settings >`

```
Answer smtrat::UnionFindModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

**Returns**

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smrat::Module](#).

**0.15.572.4.21 `checkInfSubsetForMinimality()`** `void smrat::Module::checkInfSubsetForMinimality ( std::vector< FormulaSetT >::const_iterator _infssubset,`  
`const std::string & _filename = "smaller_muses",`  
`unsigned _maxSizeDifference = 1 ) const [inherited]`

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

**Parameters**

|                                 |                                                                                                        |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>_infssubset</code>        | The infeasible subset to check for minimality.                                                         |
| <code>_filename</code>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <code>_maxSizeDifference</code> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.572.4.22 `checkModel()`** `unsigned smrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.572.4.23 `cleanModel()`** `void smrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.572.4.24 `clearLemmas()`** `void smrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.572.4.25 `clearModel()`** `void smrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.572.4.26 `clearModels()`** `void smrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.572.4.27 `clearPassedFormula()`** `void smrat::Module::clearPassedFormula ( ) [protected], [inherited]`

```
0.15.572.4.28 collectOrigins() [1/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulaSetT & _origins) const [inherited]
```

```
0.15.572.4.29 collectOrigins() [2/2] void smtrat::Module::collectOrigins (
 const FormulaT & _formula,
 FormulasT & _origins) const [inherited]
```

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

#### Parameters

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

```
0.15.572.4.30 collectTheoryPropagations() void smtrat::Module::collectTheoryPropagations ()
[inherited]
```

```
0.15.572.4.31 constraintsToInform() const carl::FastSet<FormulaT>& smtrat::Module::constraintsToInform () const [inline], [inherited]
```

#### Returns

The constraints which the backends must be informed about.

```
0.15.572.4.32 currentlySatisfied() virtual unsigned smtrat::Module::currentlySatisfied (
 const FormulaT &) const [inline], [virtual], [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smtrat::LRAbstractModule< Settings >](#), [smtrat::LRAbstractModule< LRASettingsICP >](#), and [smtrat::LRAbstractModule< LRASettings >](#)

```
0.15.572.4.33 currentlySatisfiedByBackend() unsigned smtrat::Module::currentlySatisfiedByBackend (
 const FormulaT & _formula) const [inherited]
```

#### Returns

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

```
0.15.572.4.34 deinform() void smtrat::Module::deinform (
 const FormulaT & _constraint) [inherited]
```

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.572.4.35 `deinformCore()`** `virtual void smtrat::Module::deinformCore (``const FormulaT & _constraint ) [inline], [protected], [virtual], [inherited]`

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smtrat::LRAModule< Settings >](#), [smtrat::LRAModule< LRASettingsICP >](#), and [smtrat::LRAModule< LRASettings1 >](#).

**0.15.572.4.36 `determine_smallest_origin()`** `size_t smtrat::Module::determine_smallest_origin (``const std::vector< FormulaT > & origins ) const [protected], [inherited]`**Parameters**

|                      |                              |
|----------------------|------------------------------|
| <code>origins</code> | A vector of sets of origins. |
|----------------------|------------------------------|

**Returns**

The index of the smallest (regarding the size of the sets) element of origins

**0.15.572.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::eraseSubformulaFromPassedFormula (``ModuleInput::iterator _subformula,``bool _ignoreOrigins = false ) [protected], [virtual], [inherited]`

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

**Parameters**

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

**Returns**

Reimplemented in [smtrat::ICPModule< Settings >](#), [smtrat::ICPModule< ICPSettings4 >](#), and [smtrat::ICPModule< ICPSettings1 >](#).

**0.15.572.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceivedVariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

```
0.15.572.4.39 findBestOrigin() std::vector< FormulaT >::const_iterator smtrat::Module::findBestOrigin (const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

**Parameters**

|                 |  |
|-----------------|--|
| <i>_origins</i> |  |
|-----------------|--|

**Returns**

```
0.15.572.4.40 firstSubformulaToPass() ModuleInput::const_iterator smtrat::Module::firstSubformulaToPass () const [inline], [inherited]
```

**Returns**

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

```
0.15.572.4.41 firstUncheckedReceivedSubformula() ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula () const [inline], [inherited]
```

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

```
0.15.572.4.42 freeSplittingVariable() static void smtrat::Module::freeSplittingVariable (const FormulaT & _splittingVariable) [inline], [static], [inherited]
```

```
0.15.572.4.43 generateTrivialInfeasibleSubset() void smtrat::Module::generateTrivialInfeasibleSubset () [inline], [protected], [inherited]
```

Stores the trivial infeasible subset being the set of received formulas.

```
0.15.572.4.44 getBackendsAllModels() void smtrat::Module::getBackendsAllModels () const [protected], [inherited]
```

Stores all models of a backend in the list of all models of this module.

```
0.15.572.4.45 getBackendsModel() void smtrat::Module::getBackendsModel () const [protected], [inherited]
```

```
0.15.572.4.46 getInfeasibleSubsets() [1/2] void smtrat::Module::getInfeasibleSubsets () [protected], [inherited]
```

Copies the infeasible subsets of the passed formula.

**0.15.572.4.47 getInfeasibleSubsets()** [2/2] std::vector< `FormulaSetT` > smtrat::Module::getInfeasibleSubsets (

    const `Module` & `_backend` ) const [protected], [inherited]

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.572.4.48 `getInformationRelevantFormulas()`** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`  
Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.572.4.49 `getModelEqualities()`** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`  
Partition the variables from the current model into equivalence classes according to their assigned value.  
The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

**Returns**

Equivalence classes.

**0.15.572.4.50 `getOrigins()` [1/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.572.4.51 `getOrigins()` [2/3]** `void smtrat::Module::getOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inline], [protected], [inherited]`

**Parameters**

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.572.4.52 `getOrigins()` [3/3]** `const FormulaT& smtrat::Module::getOrigins ( ModuleInput::const_iterator _formula ) const [inline], [protected], [inherited]`  
Gets the origins of the passed formula at the given position.

**Parameters**

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>_formula</i> | The position of a formula in the passed formulas. |
|-----------------|---------------------------------------------------|

**Returns**

The origins of the passed formula at the given position.

**0.15.572.4.53 `getReceivedFormulaSimplified()`** `std::pair< bool, FormulaT >` `smtrat::Module::getReceivedFormulaSimplified( )` [virtual], [inherited]

**Returns**

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.572.4.54 `hasLemmas()`** `bool smtrat::Module::hasLemmas( )` [inline], [inherited]

Checks whether this module or any of its backends provides any lemmas.

**0.15.572.4.55 `hasValidInfeasibleSubset()`** `bool smtrat::Module::hasValidInfeasibleSubset( ) const` [inherited]

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.572.4.56 `id()`** `std::size_t smtrat::Module::id( ) const` [inline], [inherited]

**Returns**

A unique ID to identify this module instance.

**0.15.572.4.57 `infeasibleSubsets()`** `const std::vector<FormulaSetT>& smtrat::Module::infeasibleSubsets( ) const` [inline], [inherited]

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.572.4.58 `inform()`** `bool smtrat::Module::inform( const FormulaT & _constraint )` [inherited]

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before `assertSubformula` is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                    |                                 |
|--------------------|---------------------------------|
| <i>_constraint</i> | The constraint to inform about. |
|--------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.572.4.59 informBackends()** `void smtrat::Module::informBackends (`

```
const FormulaT & _constraint) [inline], [protected], [inherited]
```

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.572.4.60 informCore()** `template<typename Settings >`

```
bool smtrat::UnionFindModule< Settings >::informCore (
```

```
const FormulaT & _constraint) [virtual]
```

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.572.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smtrat::Module::informedConstraints ( ) const [inline], [inherited]`**Returns**

The position of the first constraint of which no backend has been informed about.

**0.15.572.4.62 init()** `void smtrat::Module::init ( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smtrat::PBPPModule](#)< `Settings` >, [smtrat::PBGaussModule](#)< `Settings` >, [smtrat::NRAILModule](#)< `Settings` >, [smtrat::NewCoveringModule](#)< `Settings` >, [smtrat::NewCADModule](#)< `Settings` >, [smtrat::LRAModule](#)< `Settings` >, [smtrat::LRAModule](#)< `LRASettingsICP` >, [smtrat::LRAModule](#)< `LRASettings1` >, [smtrat::IntBlastModule](#)< `Settings` >, [smtrat::FPPModule](#)< `Settings` >, [smtrat::EQModule](#)< `Settings` >, [smtrat::CurryModule](#)< `Settings` >, and [smtrat::BVMModule](#)< `Settings` >.

**0.15.572.4.63 is\_minimizing()** `bool smtrat::Module::is_minimizing ( ) const [inline], [inherited]`**0.15.572.4.64 isLemmaLevel()** `bool smtrat::Module::isLemmaLevel (`

```
LemmaLevel level) [protected], [inherited]
```

Checks if current lemma level is greater or equal to given level.

**Parameters**

|              |                 |
|--------------|-----------------|
| <i>level</i> | Level to check. |
|--------------|-----------------|

**0.15.572.4.65 `isPreprocessor()`** `bool smtrat::Module::isPreprocessor () const [inline], [inherited]`**Returns**

true, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.572.4.66 `lemmas()`** `const std::vector<Lemma>& smtrat::Module::lemmas () const [inline], [inherited]`**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.572.4.67 `merge()`** `std::vector< FormulaT > smtrat::Module::merge (const std::vector< FormulaT > & _vecSetA, const std::vector< FormulaT > & _vecSetB) const [protected], [inherited]`

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                 |                                  |
|-----------------|----------------------------------|
| <i>_vecSetA</i> | A vector of sets of constraints. |
| <i>_vecSetB</i> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.572.4.68 `model()`** `const Model& smtrat::Module::model () const [inline], [inherited]`**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.572.4.69 `modelsDisjoint()`** `bool smtrat::Module::modelsDisjoint (const Model & _modelA, const Model & _modelB) [static], [protected], [inherited]`

Checks whether there is no variable assigned by both models.

**Parameters**

|                |                                |
|----------------|--------------------------------|
| <i>_modelA</i> | The first model to check for.  |
| <i>_modelB</i> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.572.4.70 `moduleName()`** template<typename Settings >  
std::string smtrat::UnionFindModule< Settings >::moduleName ( ) const [inline], [virtual]

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.572.4.71 `objective()`** carl::Variable smtrat::Module::objective ( ) const [inline], [inherited]

**0.15.572.4.72 `originInReceivedFormula()`** bool smtrat::Module::originInReceivedFormula ( const FormulaT & \_origin ) const [protected], [inherited]

**0.15.572.4.73 `passedFormulaBegin()`** ModuleInput::iterator smtrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.572.4.74 `passedFormulaEnd()`** ModuleInput::iterator smtrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]

**Returns**

An iterator to the end of the passed formula.

**0.15.572.4.75 `pPassedFormula()`** const ModuleInput\* smtrat::Module::pPassedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.572.4.76 `pReceivedFormula()`** const ModuleInput\* smtrat::Module::pReceivedFormula ( ) const [inline], [inherited]

**Returns**

A pointer to the formula passed to this module.

**0.15.572.4.77 `print()`** void smtrat::Module::print ( const std::string & \_initiation = "\*\*\*" ) const [inherited]

Prints everything relevant of the solver.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.572.4.78 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.572.4.79 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.572.4.80 printModel()** void smtrat::Module::printModel ( std::ostream & *\_out* = std::cout ) const [inherited]

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.572.4.81 printPassedFormula()** void smtrat::Module::printPassedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.572.4.82 printReceivedFormula()** void smtrat::Module::printReceivedFormula ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

```
0.15.572.4.83 probablyLooping() bool smtrat::Module::probablyLooping (
 const typename Poly::PolyType & _branchingPolynomial,
 const Rational & _branchingValue) const [protected], [inherited]
```

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

#### Returns

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

```
0.15.572.4.84 receivedFormulaChecked() void smtrat::Module::receivedFormulaChecked () [inline], [inherited]
```

Notifies that the received formulas has been checked.

```
0.15.572.4.85 receivedFormulasAsInfeasibleSubset() void smtrat::Module::receivedFormulasAs< InfeasibleSubset (
```

```
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Stores an infeasible subset consisting only of the given received formula.

```
0.15.572.4.86 receivedVariable() bool smtrat::Module::receivedVariable (
 carl::Variable::Arg _var) const [inline], [inherited]
```

```
0.15.572.4.87 remove() void smtrat::Module::remove (
 ModuleInput::const_iterator _subformula) [virtual], [inherited]
```

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

#### Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub formula of the received formula to remove. |
|--------------------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

```
0.15.572.4.88 removeCore() template<typename Settings >
void smtrat::UnionFindModule< Settings >::removeCore (
 ModuleInput::const_iterator _subformula) [virtual]
```

Removes the subformula of the received formula at the given position to the considered ones of this module.

Note that this includes every stored calculation which depended on this subformula, but should keep the other stored calculation, if possible, untouched.

#### Parameters

|                          |                                           |
|--------------------------|-------------------------------------------|
| <code>_subformula</code> | The position of the subformula to remove. |
|--------------------------|-------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.572.4.89 `removeOrigin()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigin (`

```
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

**0.15.572.4.90 `removeOrigins()`** `std::pair<ModuleInput::iterator, bool> smtrat::Module::removeOrigins (`

```
 ModuleInput::iterator _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.572.4.91 `rPassedFormula()`** `const ModuleInput& smtrat::Module::rPassedFormula ( ) const`  
`[inline], [inherited]`

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.572.4.92 `rReceivedFormula()`** `const ModuleInput& smtrat::Module::rReceivedFormula ( ) const`  
`[inline], [inherited]`

#### Returns

A reference to the formula passed to this module.

**0.15.572.4.93 `runBackends() [1/2]`** `virtual Answer smtrat::Module::runBackends ( ) [inline],`  
`[protected], [virtual], [inherited]`

Reimplemented in [smtrat::PModule](#).

**0.15.572.4.94 `runBackends() [2/2]`** `Answer smtrat::Module::runBackends (`

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                         |                                                                                                                                       |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>_final</code>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <code>_full</code>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <code>_objective</code> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.572.4.95 setId()** void smtrat::Module::setId ( std::size\_t \_id ) [inline], [inherited]  
 Sets this modules unique ID to identify itself.

#### Parameters

|                                                 |                                    |
|-------------------------------------------------|------------------------------------|
| $\leftarrow$<br>$\overline{\leftarrow}$<br>$id$ | The id to set this module's id to. |
|-------------------------------------------------|------------------------------------|

**0.15.572.4.96 setThreadPriority()** void smtrat::Module::setThreadPriority ( thread\_priority \_threadPriority ) [inline], [inherited]  
 Sets the priority of this module to get a thread for running its check procedure.

#### Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| $_threadPriority$ | The priority to set this module's thread priority to. |
|-------------------|-------------------------------------------------------|

**0.15.572.4.97 solverState()** Answer smtrat::Module::solverState ( ) const [inline], [inherited]

#### Returns

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.572.4.98 splitUnequalConstraint()** void smtrat::Module::splitUnequalConstraint ( const FormulaT & \_unequalConstraint ) [protected], [inherited]  
 Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

#### Parameters

|                      |                                                  |
|----------------------|--------------------------------------------------|
| $_unequalConstraint$ | A constraint having the relation symbol $\neq$ . |
|----------------------|--------------------------------------------------|

**0.15.572.4.99 threadPriority()** thread\_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]

#### Returns

The priority of this module to get a thread for running its check procedure.

**0.15.572.4.100 updateAllModels()** void smtrat::Module::updateAllModels ( ) [virtual], [inherited]  
 Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.  
 Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.572.4.101 updateLemmas()** void smtrat::Module::updateLemmas () [inherited]  
Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.572.4.102 updateModel()** template<typename Settings >  
void smtrat::UnionFindModule< Settings >::updateModel () const [virtual]

Updates the current assignment into the model.

Note, that this is a unique but possibly symbolic assignment maybe containing newly introduced variables.  
Reimplemented from [smtrat::Module](#).

**0.15.572.4.103 usedBackends()** const std::vector<[Module](#)\*>& smtrat::Module::usedBackends ()  
const [inline], [inherited]

#### Returns

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

### 0.15.572.5 Field Documentation

**0.15.572.5.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected],  
[inherited]

The backends of this module which have been used.

**0.15.572.5.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected],  
[inherited]

Stores all satisfying assignments.

**0.15.572.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer  
[protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.572.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform  
[protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.572.5.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.572.5.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0  
[static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.572.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smtrat::Module::mFirstSubformulaTo←  
Pass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.572.5.8 mFirstUncheckedReceivedSubformula** `ModuleInput::const_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]`  
Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.572.5.9 mFoundAnswer** `Conditionals smtrat::Module::mFoundAnswer [protected], [inherited]`  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.572.5.10 mFullCheck** `bool smtrat::Module::mFullCheck [protected], [inherited]`  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.572.5.11 mInfeasibleSubsets** `std::vector<FormulaSetT> smtrat::Module::mInfeasibleSubsets [protected], [inherited]`  
Stores the infeasible subsets.

**0.15.572.5.12 mInformedConstraints** `carl::FastSet<FormulaT> smtrat::Module::mInformedConstraints [protected], [inherited]`  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.572.5.13 mLastBranches** `std::vector<Branching> smtrat::Module::mLastBranches = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]`  
Stores the last branches in a cycle buffer.

**0.15.572.5.14 mLemmas** `std::vector<Lemma> smtrat::Module::mLemmas [protected], [inherited]`  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.572.5.15 mModel** `Model smtrat::Module::mModel [mutable], [protected], [inherited]`  
Stores the assignment of the current satisfiable result, if existent.

**0.15.572.5.16 mModelComputed** `bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]`  
True, if the model has already been computed.

**0.15.572.5.17 mNumOfBranchVarsToStore** `std::size_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]`  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.572.5.18 mObjectiveVariable** `carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]`  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.572.5.19 mOldSplittingVariables** `std::vector<FormulaT> smtrat::Module::mOldSplittingVariables [static], [inherited]`  
Reusable splitting variables.

**0.15.572.5.20 mpManager** `Manager* const smtrat::Module::mpManager [protected], [inherited]`  
A reference to the manager.

**0.15.572.5.21 mSmallerMusesCheckCounter** `unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]`  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.572.5.22 mSolverState** `std::atomic<Answer> smtrat::Module::mSolverState [protected], [inherited]`  
States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.572.5.23 mTheoryPropagations** `std::vector<TheoryPropagation> smtrat::Module::mTheoryPropagations [protected], [inherited]`

**0.15.572.5.24 mUsedBackends** `std::vector<Module*> smtrat::Module::mUsedBackends [protected], [inherited]`  
The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.572.5.25 mVariableCounters** `std::vector<std::size_t> smtrat::Module::mVariableCounters [protected], [inherited]`  
Maps variables to the number of their occurrences.

## 0.15.573 smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher Struct Reference

```
#include <SATModule.h>
```

### Public Member Functions

- `std::size_t operator() (const std::vector< Minisat::Lit > &cl) const`

#### 0.15.573.1 Member Function Documentation

**0.15.573.1.1 operator()** `template<class Settings >  
std::size_t smtrat::SATModule< Settings >::UnorderedClauseLookup::UnorderedClauseHasher<>::operator() (const std::vector< Minisat::Lit > & cl ) const [inline]`

## 0.15.574 smtrat::UnsatCore< Solver, Strategy > Class Template Reference

```
#include <UnsatCore.h>
```

### Public Member Functions

- `UnsatCore (Solver &s)`
- `void add_annotated_name (const FormulaT &formula, const std::string &name)`
- `void remove_annotated_name (const FormulaT &formula)`
- `void reset ()`
- `bool active () const`

- std::pair< Answer, FormulasT > compute\_core (const FormulasT &formulas)
- std::pair< Answer, std::vector< std::string > > compute ()

### 0.15.574.1 Constructor & Destructor Documentation

**0.15.574.1.1 UnsatCore()** template<typename Solver , UnsatCoreStrategy Strategy>  
`smtrat::UnsatCore< Solver, Strategy >::UnsatCore (`  
`Solver & s ) [inline]`

### 0.15.574.2 Member Function Documentation

**0.15.574.2.1 active()** template<typename Solver , UnsatCoreStrategy Strategy>  
`bool smtrat::UnsatCore< Solver, Strategy >::active ( ) const [inline]`

**0.15.574.2.2 add\_annotated\_name()** template<typename Solver , UnsatCoreStrategy Strategy>  
`void smtrat::UnsatCore< Solver, Strategy >::add_annotated_name (`  
`const FormulaT & formula,`  
`const std::string & name ) [inline]`

**0.15.574.2.3 compute()** template<typename Solver , UnsatCoreStrategy Strategy>  
`std::pair<Answer, std::vector<std::string> > smtrat::UnsatCore< Solver, Strategy >::compute (`  
`) [inline]`

**0.15.574.2.4 compute\_core()** template<typename Solver , UnsatCoreStrategy Strategy>  
`std::pair<Answer, FormulasT> smtrat::UnsatCore< Solver, Strategy >::compute_core (`  
`const FormulasT & formulas ) [inline]`

**0.15.574.2.5 remove\_annotated\_name()** template<typename Solver , UnsatCoreStrategy Strategy>  
`void smtrat::UnsatCore< Solver, Strategy >::remove_annotated_name (`  
`const FormulaT & formula ) [inline]`

**0.15.574.2.6 reset()** template<typename Solver , UnsatCoreStrategy Strategy>  
`void smtrat::UnsatCore< Solver, Strategy >::reset ( ) [inline]`

## 0.15.575 smtrat::unsatcore::UnsatCoreBackend< Solver, Strategy > Class Template Reference

#include <UnsatCore.h>

## 0.15.576 smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion > Class Template Reference

Implements an easy strategy to obtain an unsatisfiable core.

#include <UnsatCore\_ModelExclusion.h>

## Public Member Functions

- `UnsatCoreBackend` (`Solver &s, const FormulasT &fs)`
- `void processAssignment ()`
- `Answer compute ()`
- `std::pair< Answer, FormulasT > run ()`

### 0.15.576.1 Detailed Description

```
template<typename Solver>
class smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >
```

Implements an easy strategy to obtain an unsatisfiable core.

Essentially it computes a cover that rejects all possible models if we allow the removal of clauses.

- Allow to disable every formula  $f_i$  by encoding  $(y_i \leq f_i)$
- Find a satisfying assignment (which will set some  $f_i$  to false)
- 

### 0.15.576.2 Constructor & Destructor Documentation

```
0.15.576.2.1 UnsatCoreBackend() template<typename Solver >
smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >::UnsatCoreBackend
(
 Solver & s,
 const FormulasT & fs) [inline]
```

### 0.15.576.3 Member Function Documentation

```
0.15.576.3.1 compute() template<typename Solver >
Answer smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >::compute () [inline]
```

```
0.15.576.3.2 processAssignment() template<typename Solver >
void smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >::processAssignment () [inline]
```

```
0.15.576.3.3 run() template<typename Solver >
std::pair<Answer, FormulasT> smtrat::unsatcore::UnsatCoreBackend< Solver, UnsatCoreStrategy::ModelExclusion >::run () [inline]
```

## 0.15.577 smtrat::vs::unsignedTripleCmp Struct Reference

```
#include <State.h>
```

## Public Member Functions

- `bool operator() (UnsignedTriple n1, UnsignedTriple n2) const`

### 0.15.577.1 Member Function Documentation

```
0.15.577.1.1 operator() bool smtrat::vs::unsignedTripleCmp::operator() (
 UnsignedTriple n1,
 UnsignedTriple n2) const [inline]
```

## 0.15.578 smtrat::parser::UserFunctionInstantiator Struct Reference

```
#include <FunctionInstantiator.h>
```

### Public Member Functions

- **UserFunctionInstantiator** (const std::vector<[types::VariableType](#)> &arguments, const carl::Sort &sort, const [types::TermType](#) &definition, const std::set<[types::VariableType](#)> &auxiliaries)
- template<typename T>
 **bool convert** (const std::vector<[types::TermType](#)> &from, std::vector<T> &to) const
- template<typename T>
 **bool convert** (const std::vector<[types::TermType](#)> &from, std::vector<T> &to, [TheoryError](#) &errors) const
- virtual **bool operator()** (const std::vector<[types::TermType](#)> &, [types::TermType](#) &, [TheoryError](#) &errors) const

### Data Fields

- std::vector<[types::VariableType](#)> arguments
- carl::Sort sort
- [types::TermType](#) definition
- std::set<[types::VariableType](#)> auxiliaries
- FormulasT globalFormulas

### 0.15.578.1 Constructor & Destructor Documentation

```
0.15.578.1.1 UserFunctionInstantiator() smtrat::parser::UserFunctionInstantiator::UserFunctionInstantiator (
 const std::vector<types::VariableType> & arguments,
 const carl::Sort & sort,
 const types::TermType & definition,
 const std::set<types::VariableType> & auxiliaries) [inline]
```

### 0.15.578.2 Member Function Documentation

```
0.15.578.2.1 convert() [1/2] template<typename T>
bool smtrat::parser::FunctionInstantiator::convert (
 const std::vector<types::TermType> & from,
 std::vector<T> & to) const [inline], [inherited]
```

```
0.15.578.2.2 convert() [2/2] template<typename T>
bool smtrat::parser::FunctionInstantiator::convert (
 const std::vector<types::TermType> & from,
 std::vector<T> & to,
 TheoryError & errors) const [inline], [inherited]
```

```
0.15.578.2.3 operator() virtual bool smtrat::parser::FunctionInstantiator::operator() (
 const std::vector< types::TermType > & ,
 types::TermType & ,
 TheoryError & errors) const [inline], [virtual], [inherited]
```

### 0.15.578.3 Field Documentation

**0.15.578.3.1 arguments** std::vector<types::VariableType> smtrat::parser::UserFunctionInstantiator::arguments

**0.15.578.3.2 auxiliaries** std::set<types::VariableType> smtrat::parser::UserFunctionInstantiator::auxiliaries

**0.15.578.3.3 definition** types::TermType smtrat::parser::UserFunctionInstantiator::definition

**0.15.578.3.4 globalFormulas** FormulasT smtrat::parser::UserFunctionInstantiator::globalFormulas

**0.15.578.3.5 sort** carl::Sort smtrat::parser::UserFunctionInstantiator::sort

## 0.15.579 smtrat::validation::ValidationCollector Class Reference

```
#include <ValidationCollector.h>
```

### Public Member Functions

- ValidationPoint & **get** (const std::string &channel, const std::string &file, int line)
- const auto & **points** () const

### 0.15.579.1 Member Function Documentation

**0.15.579.1.1 get()** ValidationPoint& smtrat::validation::ValidationCollector::get (
 const std::string & channel,
 const std::string & file,
 int line ) [inline]

**0.15.579.1.2 points()** const auto& smtrat::validation::ValidationCollector::points ( ) const [inline]

## 0.15.580 smtrat::validation::ValidationPoint Class Reference

```
#include <ValidationPoint.h>
```

## Public Member Functions

- void `set_identifier` (const std::string &`channel`, const std::string &`file`, int `line`)
- const auto & `channel` () const
- auto `identifier` () const
- const auto & `formulas` () const
- std::size\_t `add` (const `FormulaT` &`f`, bool `consistent`, const std::string &`name`)
- std::size\_t `add` (const `FormulasT` &`fs`, bool `consistent`, const std::string &`name`)
- std::size\_t `add` (const `FormulaSetT` &`fss`, bool `consistent`, const std::string &`name`)
- std::size\_t `add` (const `ConstraintT` &`c`, bool `consistent`, const std::string &`name`)
- std::size\_t `add` (const `ConstraintsT` &`cs`, bool `consistent`, const std::string &`name`)

### 0.15.580.1 Member Function Documentation

**0.15.580.1.1 `add()` [1/5]** std::size\_t smtrat::validation::ValidationPoint::add (const `ConstraintsT` & `cs`, bool `consistent`, const std::string & `name`) [inline]

**0.15.580.1.2 `add()` [2/5]** std::size\_t smtrat::validation::ValidationPoint::add (const `ConstraintT` & `c`, bool `consistent`, const std::string & `name`) [inline]

**0.15.580.1.3 `add()` [3/5]** std::size\_t smtrat::validation::ValidationPoint::add (const `FormulaSetT` & `fss`, bool `consistent`, const std::string & `name`) [inline]

**0.15.580.1.4 `add()` [4/5]** std::size\_t smtrat::validation::ValidationPoint::add (const `FormulasT` & `fs`, bool `consistent`, const std::string & `name`) [inline]

**0.15.580.1.5 `add()` [5/5]** std::size\_t smtrat::validation::ValidationPoint::add (const `FormulaT` & `f`, bool `consistent`, const std::string & `name`) [inline]

**0.15.580.1.6 `channel()`** const auto& smtrat::validation::ValidationPoint::channel () const [inline]

**0.15.580.1.7 `formulas()`** const auto& smtrat::validation::ValidationPoint::formulas () const [inline]

**0.15.580.1.8 `identifier()`** auto smtrat::validation::ValidationPoint::identifier () const [inline]

```
0.15.580.1.9 set_identifier() void smtrat::validation::ValidationPoint::set_identifier (
 const std::string & channel,
 const std::string & file,
 int line) [inline]
```

## 0.15.581 smtrat::validation::ValidationPrinter< SOF > Struct Template Reference

```
#include <ValidationPrinter.h>
```

### 0.15.582 smtrat::validation::ValidationSettings Struct Reference

```
#include <ValidationSettings.h>
```

#### Public Member Functions

- bool [channel\\_active](#) (const std::string &key) const

#### Data Fields

- bool [export\\_as\\_smtp](#)
- std::string [smtp\\_filename](#)
- std::vector< std::string > [channels](#)

#### 0.15.582.1 Member Function Documentation

```
0.15.582.1.1 channel_active() bool smtrat::validation::ValidationSettings::channel_active (
 const std::string & key) const [inline]
```

#### 0.15.582.2 Field Documentation

```
0.15.582.2.1 channels std::vector<std::string> smtrat::validation::ValidationSettings::channels
```

```
0.15.582.2.2 export_as_smtp bool smtrat::validation::ValidationSettings::export_as_smtp
```

```
0.15.582.2.3 smtp_filename std::string smtrat::validation::ValidationSettings::smtp_←
filename
```

## 0.15.583 smtrat::lra::Value< T > Class Template Reference

```
#include <Value.h>
```

#### Public Member Functions

- [Value](#) ()
- [Value](#) (const T &\_num)
- [Value](#) (const T &\_num1, const T &\_num2)
- [Value](#) (const [Value](#)< T > &\_orig)
- virtual [~Value](#) ()
- [Value](#)< T > & [operator=](#) (const [Value](#)< T > &\_value)  
*Copy the content of the given value to this one.*
- [Value](#)< T > [operator+](#) (const [Value](#)< T > &\_value) const

- void `operator+=` (const `Value< T >` &`_value`)
  - `Value< T >` `operator-` (const `Value< T >` &`_value`) const
  - void `operator-=` (const `Value< T >` &`_value`)
  - `Value< T >` `operator*` (const `T &_a`) const
  - `Value< T >` `operator*=` (const `Value< T >` &`_value`) const
  - void `operator*=` (const `T &_a`)
  - `Value< T >` `operator/` (const `T &_a`) const
  - void `operator/=` (const `T &_a`)
  - bool `operator<` (const `Value< T >` &`_value`) const
  - bool `operator>` (const `Value< T >` &`_value`) const
  - bool `operator<=` (const `Value< T >` &`_value`) const
  - bool `operator>=` (const `Value< T >` &`_value`) const
  - bool `operator==` (const `Value< T >` &`_value`) const
  - bool `operator!=` (const `Value< T >` &`_value`) const
  - bool `operator==` (const `T &_a`) const
  - bool `operator!=` (const `T &_a`) const
  - bool `operator<` (const `T &_a`) const
  - bool `operator>` (const `T &_a`) const
  - bool `operator<=` (const `T &_a`) const
  - bool `operator>=` (const `T &_a`) const
  - const std::string `toString` () const
  - const `T &` `mainPart` () const
  - const `T &` `deltaPart` () const
  - const `Value< T >` & `abs_` ()
  - `Value< T >` `abs` () const
  - bool `is_zero` () const
  - `T sign` () const
- Computes the sign: 0 if main & delta part are unequal 0.*
- void `print` (std::ostream &`_out`=std::cout) const

### 0.15.583.1 Constructor & Destructor Documentation

**0.15.583.1.1 Value()** [1/4] template<class `T`>  
`smtrat::lra::Value< T >::Value ( )`

**0.15.583.1.2 Value()** [2/4] template<class `T`>  
`smtrat::lra::Value< T >::Value (`  
`const T & _num )`

#### Parameters

|                   |                      |
|-------------------|----------------------|
| <code>_num</code> | <input type="text"/> |
|-------------------|----------------------|

**0.15.583.1.3 Value()** [3/4] template<class `T`>  
`smtrat::lra::Value< T >::Value (`  
`const T & _num1,
const T & _num2 )`

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <code>_num1</code> | <input type="text"/> |
| <code>_num2</code> | <input type="text"/> |

**0.15.583.1.4 `Value()` [4/4]** template<class T >  
`smtrat::lra::Value< T >::Value (`  
    `const Value< T > & _orig )`

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <code>_orig</code> | <input type="text"/> |
|--------------------|----------------------|

**0.15.583.1.5 `~Value()`** template<class T >  
`virtual smtrat::lra::Value< T >::~Value ( ) [virtual]`

**0.15.583.2 Member Function Documentation**

**0.15.583.2.1 `abs()`** template<class T >  
`Value<T> smtrat::lra::Value< T >::abs ( ) const [inline]`

**0.15.583.2.2 `abs_()`** template<class T >  
`const Value<T>& smtrat::lra::Value< T >::abs_ ( ) [inline]`

**0.15.583.2.3 `deltaPart()`** template<class T >  
`const T& smtrat::lra::Value< T >::deltaPart ( ) const [inline]`

**Returns**

**0.15.583.2.4 `is_zero()`** template<class T >  
`bool smtrat::lra::Value< T >::is_zero ( ) const [inline]`

**0.15.583.2.5 `mainPart()`** template<class T >  
`const T& smtrat::lra::Value< T >::mainPart ( ) const [inline]`

**Returns**

**0.15.583.2.6 `operator"!="() [1/2]`** template<class T >  
`bool smtrat::lra::Value< T >::operator!= (`  
    `const T & _a ) const [inline]`

```
0.15.583.2.7 operator"!=() [2/2] template<class T >
bool smtrat::lra::Value< T >::operator!= (
 const Value< T > & _value) const [inline]
```

```
0.15.583.2.8 operator*() [1/2] template<class T >
Value<T> smtrat::lra::Value< T >::operator* (
 const T & _a) const
```

**Parameters**

|          |  |
|----------|--|
| ↔        |  |
| ↔        |  |
| <i>a</i> |  |

**Returns**

```
0.15.583.2.9 operator*() [2/2] template<class T >
Value<T> smtrat::lra::Value< T >::operator* (
 const Value< T > & _value) const
```

**Parameters**

|        |  |
|--------|--|
| _value |  |
|--------|--|

```
0.15.583.2.10 operator*=(()) [1/2] template<class T >
void smtrat::lra::Value< T >::operator*=
(
 const T & _a)
```

**Parameters**

|          |  |
|----------|--|
| ↔        |  |
| ↔        |  |
| <i>a</i> |  |

```
0.15.583.2.11 operator*=(()) [2/2] template<class T >
void smtrat::lra::Value< T >::operator*=
(
 const Value< T > & _value)
```

**Parameters**

|        |  |
|--------|--|
| _value |  |
|--------|--|

```
0.15.583.2.12 operator+() template<class T >
Value<T> smtrat::lra::Value< T >::operator+
(
 const Value< T > & _value) const
```

**Parameters**

|                     |                          |
|---------------------|--------------------------|
| <code>_value</code> | <input type="checkbox"/> |
|---------------------|--------------------------|

**Returns**

**0.15.583.2.13 `operator+=()`** template<class T >  
void `smtrat::lra::Value< T >::operator+=` (  
    const `Value< T >` & `_value` )

**Parameters**

|                     |                          |
|---------------------|--------------------------|
| <code>_value</code> | <input type="checkbox"/> |
|---------------------|--------------------------|

**0.15.583.2.14 `operator-()`** template<class T >  
`Value<T>` `smtrat::lra::Value< T >::operator-` (  
    const `Value< T >` & `_value` ) const

**Parameters**

|                     |                          |
|---------------------|--------------------------|
| <code>_value</code> | <input type="checkbox"/> |
|---------------------|--------------------------|

**Returns**

**0.15.583.2.15 `operator-=()`** template<class T >  
void `smtrat::lra::Value< T >::operator-=` (  
    const `Value< T >` & `_value` )

**Parameters**

|                     |                          |
|---------------------|--------------------------|
| <code>_value</code> | <input type="checkbox"/> |
|---------------------|--------------------------|

**0.15.583.2.16 `operator()`** template<class T >  
`Value<T>` `smtrat::lra::Value< T >::operator/` (  
    const `T` & `_a` ) const

**Parameters**

|                               |                          |
|-------------------------------|--------------------------|
| $\leftrightarrow$             | <input type="checkbox"/> |
| $\underline{\leftrightarrow}$ | <input type="checkbox"/> |
| $a$                           | <input type="checkbox"/> |

Returns

**0.15.583.2.17 operator/=( )** template<class T >  
void smtrat::lra::Value< T >::operator/= ( const T & \_a )

Parameters

|                      |  |
|----------------------|--|
| $\leftrightarrow$    |  |
| $\_ \leftrightarrow$ |  |
| a                    |  |

**0.15.583.2.18 operator<() [1/2]** template<class T >  
bool smtrat::lra::Value< T >::operator< ( const T & \_a ) const

Parameters

|                      |  |
|----------------------|--|
| $\leftrightarrow$    |  |
| $\_ \leftrightarrow$ |  |
| a                    |  |

Returns

**0.15.583.2.19 operator<() [2/2]** template<class T >  
bool smtrat::lra::Value< T >::operator< ( const Value< T > & \_value ) const

Parameters

|           |  |
|-----------|--|
| $\_value$ |  |
|           |  |

Returns

**0.15.583.2.20 operator<=() [1/2]** template<class T >  
bool smtrat::lra::Value< T >::operator<= ( const T & \_a ) const

Parameters

|                      |  |
|----------------------|--|
| $\leftrightarrow$    |  |
| $\_ \leftrightarrow$ |  |
| a                    |  |

**Returns**

**0.15.583.2.21 operator<=()** [2/2] template<class T >  
bool smtrat::lra::Value< T >::operator<= (  
 const Value< T > & \_value ) const

**Parameters**

|        |  |
|--------|--|
| _value |  |
|--------|--|

**Returns**

**0.15.583.2.22 operator=()** template<class T >  
Value<T>& smtrat::lra::Value< T >::operator= (  
 const Value< T > & \_value )

Copy the content of the given value to this one.

**Parameters**

|        |                    |
|--------|--------------------|
| _value | The value to copy. |
|--------|--------------------|

**Returns**

This value after copying.

**0.15.583.2.23 operator==()** [1/2] template<class T >  
bool smtrat::lra::Value< T >::operator== (  
 const T & \_a ) const

**Parameters**

|   |  |
|---|--|
| ↔ |  |
| ↔ |  |
| a |  |

**Returns**

**0.15.583.2.24 operator==()** [2/2] template<class T >  
bool smtrat::lra::Value< T >::operator== (  
 const Value< T > & \_value ) const

**Parameters**

|        |  |
|--------|--|
| _value |  |
|--------|--|

Returns

**0.15.583.2.25 operator>() [1/2]** template<class T >  
bool smtrat::lra::Value< T >::operator> (  
 const T & \_a ) const

Parameters

|   |  |
|---|--|
| ↔ |  |
| ↔ |  |
| a |  |

Returns

**0.15.583.2.26 operator>() [2/2]** template<class T >  
bool smtrat::lra::Value< T >::operator> (  
 const Value< T > & \_value ) const [inline]

**0.15.583.2.27 operator>=()** [1/2] template<class T >  
bool smtrat::lra::Value< T >::operator>= (  
 const T & \_a ) const

Parameters

|   |  |
|---|--|
| ↔ |  |
| ↔ |  |
| a |  |

Returns

**0.15.583.2.28 operator>=()** [2/2] template<class T >  
bool smtrat::lra::Value< T >::operator>= (  
 const Value< T > & \_value ) const [inline]

**0.15.583.2.29 print()** template<class T >  
void smtrat::lra::Value< T >::print (  
 std::ostream & \_out = std::cout ) const

Parameters

|      |  |
|------|--|
| _out |  |
|------|--|

**0.15.583.2.30 sign()** template<class T>  
T smrat::lra::Value< T >::sign () const  
Computes the sign: 0 if main & delta part are unequal 0.  
Else +/- 1 determined by the sign of main part (if set) else delta part.

**0.15.583.2.31 toString()** template<class T>  
const std::string smrat::lra::Value< T >::toString () const

Returns

## 0.15.584 smrat::lra::Variable< T1, T2 > Class Template Reference

#include <Variable.h>

### Public Member Functions

- **Variable** (size\_t \_position, const typename Poly::PolyType \*\_expression, ModuleInput::iterator \_defaultBoundPosition, bool \_isInteger, size\_t \_id)
- **Variable** (typename std::list< std::list< std::pair< Variable< T1, T2 > \*, T2 >>>::iterator \_positionInNonActives, const typename Poly::PolyType \*\_expression, ModuleInput::iterator \_defaultBoundPosition, bool \_isInteger, size\_t \_id)
- virtual ~Variable ()
- const Value< T1 > & **assignment** () const
- Value< T1 > & **rAssignment** ()
- void **storeAssignment** ()
- const Value< T1 > & **lastConsistentAssignment** () const
- void **setBasic** (bool \_basic)
- bool **isBasic** () const
- bool **isOriginal** () const
- bool **is\_integer** () const
- bool **hasBound** () const
- bool **involvesEquation** () const
- EntryID **startEntry** () const
- EntryID & **rStartEntry** ()
- size\_t **size** () const
- size\_t & **rSize** ()
- double **conflictActivity** () const
- void **setSupremum** (const Bound< T1, T2 > \*\_supremum)
- const Bound< T1, T2 > \* **pSupremum** () const
- const Bound< T1, T2 > & **supremum** () const
- void **setInfimum** (const Bound< T1, T2 > \*\_infimum)
- const Bound< T1, T2 > \* **pInfimum** () const
- const Bound< T1, T2 > & **infimum** () const
- size\_t **position** () const
- size\_t **id** () const
- void **setPosition** (size\_t \_position)
- bool **isConflicting** () const
- std::list< std::list< std::pair< Variable< T1, T2 > \*, T2 >>>::iterator **positionInNonActives** () const
- void **setPositionInNonActives** (typename std::list< std::list< std::pair< Variable< T1, T2 > \*, T2 >>>::iterator \_positionInNonActives)
- size\_t **rLowerBoundsSize** ()
- size\_t **rUpperBoundsSize** ()
- const Bound< T1, T2 >::BoundSet & **upperbounds** () const
- const Bound< T1, T2 >::BoundSet & **lowerbounds** () const
- Bound< T1, T2 >::BoundSet & **rUpperbounds** ()

- `Bound< T1, T2 >::BoundSet & rLowerbounds ()`
- `size_t & rPosition ()`
- `const Poly::PolyType * pExpression () const`
- `const Poly::PolyType & expression () const`
- `const T2 & factor () const`
- `T2 & rFactor ()`
- `unsigned isSatisfiedBy (const RationalAssignment &_ass) const`
- `FormulaT inConflictWith (const RationalAssignment &_ass) const`
- `void updateConflictActivity ()`
- `std::pair< const Bound< T1, T2 > *, bool > addUpperBound (Value< T1 > *const _val, ModuleInput::iterator _position, const FormulaT &_constraint, bool _deduced=false)`
- `std::pair< const Bound< T1, T2 > *, bool > addLowerBound (Value< T1 > *const _val, ModuleInput::iterator _position, const FormulaT &_constraint, bool _deduced=false)`
- `std::pair< const Bound< T1, T2 > *, bool > addEqualBound (Value< T1 > *const _val, ModuleInput::iterator _position, const FormulaT &_constraint)`
- `void removeBound (const Bound< T1, T2 > *_bound)`
- `bool deactivateBound (const Bound< T1, T2 > *bound, ModuleInput::iterator _position)`
- `RationalInterval getVariableBounds () const`
- `FormulasT getDefiningOrigins () const`
- `bool operator< (const Variable &_variable) const`
- `bool operator> (const Variable &_variable) const`
- `bool operator== (const Variable &_variable) const`
- `bool operator!= (const Variable &_variable) const`
- `void print (std::ostream &_out=std::cout) const`
- `void printAllBounds (std::ostream &_out=std::cout, const std::string _init="") const`

#### 0.15.584.1 Constructor & Destructor Documentation

```
0.15.584.1.1 Variable() [1/2] template<typename T1 , typename T2 >
smrat::lra::Variable< T1, T2 >::Variable (
 size_t _position,
 const typename Poly::PolyType * _expression,
 ModuleInput::iterator _defaultBoundPosition,
 bool _isInteger,
 size_t _id)
```

##### Parameters

|                                    |  |
|------------------------------------|--|
| <code>_position</code>             |  |
| <code>_expression</code>           |  |
| <code>_defaultBoundPosition</code> |  |
| <code>_isInteger</code>            |  |
| <code>_id</code>                   |  |

```
0.15.584.1.2 Variable() [2/2] template<typename T1 , typename T2 >
smrat::lra::Variable< T1, T2 >::Variable (
 typename std::list< std::list< std::pair< Variable< T1, T2 > *, T2 >>>::iterator
 _positionInNonActives,
 const typename Poly::PolyType * _expression,
 ModuleInput::iterator _defaultBoundPosition,
 bool _isInteger,
 size_t _id)
```

**Parameters**

|                                    |  |
|------------------------------------|--|
| <code>_positionInNonActives</code> |  |
| <code>_expression</code>           |  |
| <code>_defaultBoundPosition</code> |  |
| <code>_isInteger</code>            |  |
| <code>_id</code>                   |  |

**0.15.584.1.3 ~Variable()** template<typename T1 , typename T2 >  
virtual smtrat::lra::Variable< T1, T2 >::~Variable ( ) [virtual]

**0.15.584.2 Member Function Documentation**

**0.15.584.2.1 addEqualBound()** template<typename T1 , typename T2 >  
std::pair<const Bound<T1, T2>\*, bool> smtrat::lra::Variable< T1, T2 >::addEqualBound (   
    Value< T1 > \*const \_val,  
    ModuleInput::iterator \_position,  
    const FormulaT & \_constraint )

**Parameters**

|                          |  |
|--------------------------|--|
| <code>_val</code>        |  |
| <code>_position</code>   |  |
| <code>_constraint</code> |  |

**Returns**

**0.15.584.2.2 addLowerBound()** template<typename T1 , typename T2 >  
std::pair<const Bound<T1, T2>\*, bool> smtrat::lra::Variable< T1, T2 >::addLowerBound (   
    Value< T1 > \*const \_val,  
    ModuleInput::iterator \_position,  
    const FormulaT & \_constraint,  
    bool \_deduced = false )

**Parameters**

|                          |  |
|--------------------------|--|
| <code>_val</code>        |  |
| <code>_position</code>   |  |
| <code>_constraint</code> |  |
| <code>_deduced</code>    |  |

**Returns**

**0.15.584.2.3 addUpperBound()** template<typename T1 , typename T2 >  
std::pair<const Bound<T1, T2>\*, bool> smtrat::lra::Variable< T1, T2 >::addUpperBound (

```
Value< T1 > *const _val,
ModuleInput::iterator _position,
const FormulaT & _constraint,
bool _deduced = false)
```

**Parameters**

|                    |  |
|--------------------|--|
| <i>_val</i>        |  |
| <i>_position</i>   |  |
| <i>_constraint</i> |  |
| <i>_deduced</i>    |  |

**Returns**

**0.15.584.2.4 assignment()** template<typename T1 , typename T2 >  
 const Value<T1>& smtrat::lra::Variable< T1, T2 >::assignment () const [inline]

**Returns**

**0.15.584.2.5 conflictActivity()** template<typename T1 , typename T2 >  
 double smtrat::lra::Variable< T1, T2 >::conflictActivity () const [inline]

**Returns**

**0.15.584.2.6 deactivateBound()** template<typename T1 , typename T2 >  
 bool smtrat::lra::Variable< T1, T2 >::deactivateBound (  
     const Bound< T1, T2 > \* bound,  
     ModuleInput::iterator \_position )

**Parameters**

|                  |  |
|------------------|--|
| <i>bound</i>     |  |
| <i>_position</i> |  |

**Returns**

**0.15.584.2.7 expression()** template<typename T1 , typename T2 >  
 const PolyType& smtrat::lra::Variable< T1, T2 >::expression () const [inline]

**Returns**

**0.15.584.2.8 `factor()`** template<typename T1 , typename T2 >  
const T2& smtrat::lra::Variable< T1, T2 >::factor ( ) const [inline]

Returns

**0.15.584.2.9 `getDefiningOrigins()`** template<typename T1 , typename T2 >  
FormulasT smtrat::lra::Variable< T1, T2 >::getDefiningOrigins ( ) const

Returns

**0.15.584.2.10 `getVariableBounds()`** template<typename T1 , typename T2 >  
RationalInterval smtrat::lra::Variable< T1, T2 >::getVariableBounds ( ) const

Returns

**0.15.584.2.11 `hasBound()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::hasBound ( ) const [inline]

Returns

**0.15.584.2.12 `id()`** template<typename T1 , typename T2 >  
size\_t smtrat::lra::Variable< T1, T2 >::id ( ) const [inline]

**0.15.584.2.13 `inConflictWith()`** template<typename T1 , typename T2 >  
FormulaT smtrat::lra::Variable< T1, T2 >::inConflictWith (   
const RationalAssignment & \_ass ) const [inline]

Parameters

|      |                                 |
|------|---------------------------------|
| _ass | <input type="button" value=""/> |
|------|---------------------------------|

Returns

**0.15.584.2.14 `infimum()`** template<typename T1 , typename T2 >  
const Bound<T1, T2>& smtrat::lra::Variable< T1, T2 >::infimum ( ) const [inline]

Returns

**0.15.584.2.15 `involvesEquation()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::involvesEquation ( ) const [inline]

Returns

**0.15.584.2.16 `is_integer()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::is\_integer ( ) const [inline]

Returns

**0.15.584.2.17 `isBasic()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::isBasic ( ) const [inline]

Returns

**0.15.584.2.18 `isConflicting()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::isConflicting ( ) const [inline]

**0.15.584.2.19 `isOriginal()`** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::isOriginal ( ) const [inline]

Returns

**0.15.584.2.20 `isSatisfiedBy()`** template<typename T1 , typename T2 >  
unsigned smtrat::lra::Variable< T1, T2 >::isSatisfiedBy (   
const RationalAssignment & \_ass ) const [inline]

Parameters

|      |  |
|------|--|
| _ass |  |
|------|--|

Returns

**0.15.584.2.21 `lastConsistentAssignment()`** template<typename T1 , typename T2 >  
const Value<T1>& smtrat::lra::Variable< T1, T2 >::lastConsistentAssignment ( ) const [inline]

Returns

**0.15.584.2.22 lowerbounds()** template<typename T1 , typename T2 >  
const Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::lowerbounds ( ) const [inline]

Returns

**0.15.584.2.23 operator"!=()** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::operator!= (   
const Variable< T1, T2 > & \_variable ) const [inline]

**0.15.584.2.24 operator<()** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::operator< (   
const Variable< T1, T2 > & \_variable ) const [inline]

Parameters

|               |
|---------------|
| <i>_bound</i> |
|---------------|

Returns

**0.15.584.2.25 operator==()** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::operator== (   
const Variable< T1, T2 > & \_variable ) const [inline]

**0.15.584.2.26 operator>()** template<typename T1 , typename T2 >  
bool smtrat::lra::Variable< T1, T2 >::operator> (   
const Variable< T1, T2 > & \_variable ) const [inline]

**0.15.584.2.27 pExpression()** template<typename T1 , typename T2 >  
const Poly::PolyType\* smtrat::lra::Variable< T1, T2 >::pExpression ( ) const [inline]

Returns

**0.15.584.2.28 plInfimum()** template<typename T1 , typename T2 >  
const Bound<T1, T2>\* smtrat::lra::Variable< T1, T2 >::plInfimum ( ) const [inline]

Returns

**0.15.584.2.29 position()** template<typename T1 , typename T2 >  
size\_t smtrat::lra::Variable< T1, T2 >::position ( ) const [inline]

Returns

```
0.15.584.2.30 positionInNonActives() template<typename T1 , typename T2 >
std::list<std::list<std::pair<Variable<T1,T2>*,T2> >::iterator smtrat::lra::Variable< T1,
T2 >::positionInNonActives () const [inline]
```

Returns

```
0.15.584.2.31 print() template<typename T1 , typename T2 >
void smtrat::lra::Variable< T1, T2 >::print (
 std::ostream & _out = std::cout) const
```

Parameters

|                   |  |
|-------------------|--|
| <code>_out</code> |  |
|-------------------|--|

```
0.15.584.2.32 printAllBounds() template<typename T1 , typename T2 >
void smtrat::lra::Variable< T1, T2 >::printAllBounds (
 std::ostream & _out = std::cout,
 const std::string _init = "") const
```

Parameters

|                    |  |
|--------------------|--|
| <code>_out</code>  |  |
| <code>_init</code> |  |

```
0.15.584.2.33 pSupremum() template<typename T1 , typename T2 >
const Bound<T1, T2>* smtrat::lra::Variable< T1, T2 >::pSupremum () const [inline]
```

Returns

```
0.15.584.2.34 rAssignment() template<typename T1 , typename T2 >
Value<T1>& smtrat::lra::Variable< T1, T2 >::rAssignment () [inline]
```

Returns

```
0.15.584.2.35 removeBound() template<typename T1 , typename T2 >
void smtrat::lra::Variable< T1, T2 >::removeBound (
 const Bound< T1, T2 > * _bound)
```

Parameters

|                     |  |
|---------------------|--|
| <code>_bound</code> |  |
|---------------------|--|

**0.15.584.2.36 rFactor()** template<typename T1 , typename T2 >  
T2& smtrat::lra::Variable< T1, T2 >::rFactor ( ) [inline]

Returns

**0.15.584.2.37 rLowerbounds()** template<typename T1 , typename T2 >  
Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::rLowerbounds ( ) [inline]

Returns

**0.15.584.2.38 rLowerBoundsSize()** template<typename T1 , typename T2 >  
size\_t smtrat::lra::Variable< T1, T2 >::rLowerBoundsSize ( ) [inline]

Returns

**0.15.584.2.39 rPosition()** template<typename T1 , typename T2 >  
size\_t& smtrat::lra::Variable< T1, T2 >::rPosition ( ) [inline]

Returns

**0.15.584.2.40 rSize()** template<typename T1 , typename T2 >  
size\_t& smtrat::lra::Variable< T1, T2 >::rSize ( ) [inline]

Returns

**0.15.584.2.41 rStartEntry()** template<typename T1 , typename T2 >  
EntryID& smtrat::lra::Variable< T1, T2 >::rStartEntry ( ) [inline]

Returns

**0.15.584.2.42 rUpperbounds()** template<typename T1 , typename T2 >  
Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::rUpperbounds ( ) [inline]

Returns

**0.15.584.2.43 rUpperBoundsSize()** template<typename T1 , typename T2 >  
size\_t smtrat::lra::Variable< T1, T2 >::rUpperBoundsSize ( ) [inline]

Returns

**0.15.584.2.44 setBasic()** template<typename T1 , typename T2 >  
void smtrat::lra::Variable< T1, T2 >::setBasic (   
    bool \_basic ) [inline]

Parameters

|               |
|---------------|
| <i>_basic</i> |
|---------------|

**0.15.584.2.45 setInfimum()** template<typename T1 , typename T2 >  
void smtrat::lra::Variable< T1, T2 >::setInfimum (   
    const Bound< T1, T2 > \* \_infimum ) [inline]

Parameters

|                 |
|-----------------|
| <i>_infimum</i> |
|-----------------|

**0.15.584.2.46 setPosition()** template<typename T1 , typename T2 >  
void smtrat::lra::Variable< T1, T2 >::setPosition (   
    size\_t \_position ) [inline]

Parameters

|                  |
|------------------|
| <i>_position</i> |
|------------------|

**0.15.584.2.47 setPositionInNonActives()** template<typename T1 , typename T2 >  
void smtrat::lra::Variable< T1, T2 >::setPositionInNonActives (   
    typename std::list< std::list< std::pair< Variable< T1, T2 > \*, T2 >>>::iterator  
\_positionInNonActives ) [inline]

Parameters

|                              |
|------------------------------|
| <i>_positionInNonActives</i> |
|------------------------------|

**0.15.584.2.48 setSupremum()** template<typename T1 , typename T2 >  
void smtrat::lra::Variable< T1, T2 >::setSupremum (   
    const Bound< T1, T2 > \* \_supremum ) [inline]

Parameters

|                  |
|------------------|
| <i>_supremum</i> |
|------------------|

**0.15.584.2.49 `size()`** template<typename T1 , typename T2 >  
size\_t smtrat::lra::Variable< T1, T2 >::size ( ) const [inline]

Returns

**0.15.584.2.50 `startEntry()`** template<typename T1 , typename T2 >  
EntryID smtrat::lra::Variable< T1, T2 >::startEntry ( ) const [inline]

Returns

**0.15.584.2.51 `storeAssignment()`** template<typename T1 , typename T2 >  
void smtrat::lra::Variable< T1, T2 >::storeAssignment ( ) [inline]

**0.15.584.2.52 `supremum()`** template<typename T1 , typename T2 >  
const Bound<T1, T2>& smtrat::lra::Variable< T1, T2 >::supremum ( ) const [inline]

Returns

**0.15.584.2.53 `updateConflictActivity()`** template<typename T1 , typename T2 >  
void smtrat::lra::Variable< T1, T2 >::updateConflictActivity ( ) [inline]

**0.15.584.2.54 `upperbounds()`** template<typename T1 , typename T2 >  
const Bound<T1, T2>::BoundSet& smtrat::lra::Variable< T1, T2 >::upperbounds ( ) const [inline]

Returns

## 0.15.585 smtrat::vb::Variable< T > Class Template Reference

Class for a variable.

```
#include <VariableBounds.h>
```

### Public Member Functions

- `Variable ()`  
*Constructs this variable.*
- `~Variable ()`
- `bool conflicting () const`
- `const Bound< T >* addBound (const ConstraintT &_constraint, const carl::Variable &_var, const T &_origin)`  
*Adds the bound corresponding to the constraint to the given variable.*
- `bool updateBounds (const Bound< T > &_changedBound)`  
*Updates the infimum and supremum of this variable.*
- `bool updatedExactInterval () const`

- void `exactIntervalHasBeenUpdated () const`  
*Sets the flag indicating that the stored exact interval representing the variable's bounds is up to date to false.*
- bool `updatedDoubleInterval () const`
- void `doubleIntervalHasBeenUpdated () const`  
*Sets the flag indicating that the stored double interval representing the variable's bounds is up to date to false.*
- const `Bound< T > * pSupremum () const`
- const `Bound< T > & supremum () const`
- const `Bound< T > * plnfimum () const`
- const `Bound< T > & infimum () const`
- const `BoundSet & upperbounds () const`
- const `BoundSet & lowerbounds () const`

### 0.15.585.1 Detailed Description

```
template<typename T>
class smrat::vb::Variable< T >
```

Class for a variable.

### 0.15.585.2 Constructor & Destructor Documentation

**0.15.585.2.1 Variable()** template<typename T >  
`smrat::vb::Variable< T >::Variable ( )`  
Constructs this variable.

**0.15.585.2.2 ~Variable()** template<typename T >  
`smrat::vb::Variable< T >::~Variable ( )`

### 0.15.585.3 Member Function Documentation

**0.15.585.3.1 addBound()** template<typename T >  
`const Bound<T>* smrat::vb::Variable< T >::addBound (`  
`const ConstraintT & _constraint,`  
`const carl::Variable< T > & _var,`  
`const T & _origin )`

Adds the bound corresponding to the constraint to the given variable.

The constraint is expected to contain only one variable and this one only linearly.

#### Parameters

|                          |                                                                                                                                  |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <code>_constraint</code> | A pointer to a constraint of the form $ax \sim b$ .                                                                              |
| <code>_var</code>        | Hence, the variable $x$ .                                                                                                        |
| <code>_origin</code>     | The origin of the constraint. This could be the constraint itself or anything else in the data structure which uses this object. |
| <code>_limit</code>      |                                                                                                                                  |

#### Returns

The added bound.

**0.15.585.3.2 conflicting()** template<typename T >  
bool `smtrat::vb::Variable`< T >::conflicting () const  
**Returns**

true, if there is a conflict; false, otherwise.

**0.15.585.3.3 doubleIntervalHasBeenUpdated()** template<typename T >  
void `smtrat::vb::Variable`< T >::doubleIntervalHasBeenUpdated () const [inline]  
Sets the flag indicating that the stored double interval representing the variable's bounds is up to date to false.

**0.15.585.3.4 exactIntervalHasBeenUpdated()** template<typename T >  
void `smtrat::vb::Variable`< T >::exactIntervalHasBeenUpdated () const [inline]  
Sets the flag indicating that the stored exact interval representing the variable's bounds is up to date to false.

**0.15.585.3.5 infimum()** template<typename T >  
const `Bound`<T>& `smtrat::vb::Variable`< T >::infimum () const [inline]  
**Returns**

A constant reference to the infimum of this variable.

**0.15.585.3.6 lowerbounds()** template<typename T >  
const `BoundSet`& `smtrat::vb::Variable`< T >::lowerbounds () const [inline]  
**Returns**

A constant reference to the set of lower bounds of this variable.

**0.15.585.3.7 pInfimum()** template<typename T >  
const `Bound`<T>\* `smtrat::vb::Variable`< T >::pInfimum () const [inline]  
**Returns**

A pointer to the infimum of this variable.

**0.15.585.3.8 pSupremum()** template<typename T >  
const `Bound`<T>\* `smtrat::vb::Variable`< T >::pSupremum () const [inline]  
**Returns**

A pointer to the supremum of this variable.

**0.15.585.3.9 supremum()** template<typename T >  
const `Bound`<T>& `smtrat::vb::Variable`< T >::supremum () const [inline]  
**Returns**

A constant reference to the supremum of this variable.

**0.15.585.3.10 updateBounds()** template<typename T >  
bool `smtrat::vb::Variable`< T >::updateBounds (  
    const `Bound`< T > & `_changedBound` )  
Updates the infimum and supremum of this variable.

**Parameters**

|                            |                                                                        |
|----------------------------|------------------------------------------------------------------------|
| <code>_changedBound</code> | The bound, for which we certainly know that it got deactivated before. |
|----------------------------|------------------------------------------------------------------------|

**Returns**

true, if there is a conflict; false, otherwise.

```
0.15.585.3.11 updatedDoubleInterval() template<typename T >
bool smtrat::vb::Variable< T >::updatedDoubleInterval () const [inline]
```

**Returns**

true, if the stored double interval representing the variable's bounds is up to date. false, otherwise

```
0.15.585.3.12 updatedExactInterval() template<typename T >
bool smtrat::vb::Variable< T >::updatedExactInterval () const [inline]
```

**Returns**

true, if the stored exact interval representing the variable's bounds is up to date. false, otherwise

```
0.15.585.3.13 upperbounds() template<typename T >
const BoundSet& smtrat::vb::Variable< T >::upperbounds () const [inline]
```

**Returns**

A constant reference to the set of upper bounds of this variable.

## 0.15.586 smtrat::vb::VariableBounds< T > Class Template Reference

Class to manage the bounds of a variable.

```
#include <VariableBounds.h>
```

### Public Types

- **typedef std::map< ConstraintT, const Bound< T > \* > ConstraintBoundMap**  
*A map from Constraint pointers to Bound pointers.*
- **typedef carl::FastMap< carl::Variable, Variable< T > \* > VariableMap**  
*A hash-map from arithmetic variables to variables managing the bounds.*

### Public Member Functions

- **VariableBounds ()**  
*Constructs a variable bounds manager.*
- **~VariableBounds ()**  
*Destructs a variable bounds manager.*
- **void clear ()**
- **bool empty () const**
- **bool addBound (const ConstraintT &\_constraint, const T &\_origin)**  
*Updates the variable bounds by the given constraint.*
- **bool addBound (const FormulaT &\_formula, const T &\_origin)**
- **unsigned removeBound (const ConstraintT &\_constraint, const T &\_origin)**

- *Removes all effects the given constraint has on the variable bounds.*
- unsigned `removeBound` (const `FormulaT` &\_formula, const `T` &\_origin)
- unsigned `removeBound` (const `ConstraintT` &\_constraint, const `T` &\_origin, carl::Variable \*&\_changed)
- *Removes all effects the given constraint has on the variable bounds.*
- const `EvalRationalIntervalMap` & `getEvalIntervalMap` () const
- Creates an evalintervalmap corresponding to the variable bounds.*
- const `RationalInterval` & `getInterval` (const carl::Variable &\_var) const
- Creates an interval corresponding to the variable bounds of the given variable.*
- `RationalInterval` `getInterval` (carl::Monomial::Arg \_mon) const
- Creates an interval corresponding to the bounds of the given monomial.*
- `RationalInterval` `getInterval` (const `TermT` &\_term) const
- Creates an interval corresponding to the bounds of the given term.*
- const `smrat::EvalDoubleIntervalMap` & `getIntervalMap` () const
- Creates an interval map corresponding to the variable bounds.*
- const carl::Interval< double > & `getDoubleInterval` (const carl::Variable &\_var) const
- Creates a double interval corresponding to the variable bounds of the given variable.*
- std::vector< `T` > `getOriginsOfBounds` (const carl::Variable &\_var) const
- std::set< `T` > `getOriginSetOfBounds` (const carl::Variable &\_var) const
- std::vector< `T` > `getOriginsOfBounds` (const carl::Variables &\_variables) const
- std::set< `T` > `getOriginSetOfBounds` (const carl::Variables &\_variables) const
- std::vector< `T` > `getOriginsOfBounds` () const
- Collect the origins to the supremums and infimums of all variables.*
- std::set< `T` > `getOriginSetOfBounds` () const
- std::vector< std::pair< std::vector< `ConstraintT` >, `ConstraintT` > > `getBoundDeductions` () const
- void `print` (std::ostream &\_out=std::cout, const std::string \_init="", bool \_printAllBounds=false) const
- Prints the variable bounds.*
- bool `isConflicting` () const
- std::set< `T` > `getConflict` () const

### 0.15.586.1 Detailed Description

```
template<typename T>
class smrat::vb::VariableBounds< T >
```

Class to manage the bounds of a variable.

### 0.15.586.2 Member Typedef Documentation

**0.15.586.2.1 ConstraintBoundMap** template<typename `T` >  
 typedef std::map<`ConstraintT`, const `Bound<T>*> smrat::vb::VariableBounds< T >::ConstraintBoundMap  
 A map from Constraint pointers to Bound pointers.`

**0.15.586.2.2 VariableMap** template<typename `T` >  
 typedef carl::FastMap<carl::Variable, `Variable<T>*> smrat::vb::VariableBounds< T >::VariableMap  
 A hash-map from arithmetic variables to variables managing the bounds.`

### 0.15.586.3 Constructor & Destructor Documentation

**0.15.586.3.1 VariableBounds()** template<typename T >  
`smtrat::vb::VariableBounds< T >::VariableBounds ( )`  
Constructs a variable bounds manager.

**0.15.586.3.2 ~VariableBounds()** template<typename T >  
`smtrat::vb::VariableBounds< T >::~VariableBounds ( )`  
Destructs a variable bounds manager.

#### 0.15.586.4 Member Function Documentation

**0.15.586.4.1 addBound() [1/2]** template<typename T >  
`bool smtrat::vb::VariableBounds< T >::addBound (`  
`const ConstraintT & _constraint,`  
`const T & _origin )`

Updates the variable bounds by the given constraint.

##### Parameters

|                          |                                     |
|--------------------------|-------------------------------------|
| <code>_constraint</code> | The constraint to consider.         |
| <code>_origin</code>     | The origin of the given constraint. |

##### Returns

true, if the variable bounds have been changed; false, otherwise.

**0.15.586.4.2 addBound() [2/2]** template<typename T >  
`bool smtrat::vb::VariableBounds< T >::addBound (`  
`const FormulaT & _formula,`  
`const T & _origin )`

**0.15.586.4.3 clear()** template<typename T >  
`void smtrat::vb::VariableBounds< T >::clear ( )`

**0.15.586.4.4 empty()** template<typename T >  
`bool smtrat::vb::VariableBounds< T >::empty ( ) const [inline]`

**0.15.586.4.5 getBoundDeductions()** template<typename T >  
`std::vector<std::pair<std::vector<ConstraintT>, ConstraintT> > smtrat::vb::VariableBounds< T >::getBoundDeductions ( ) const`

##### Returns

The deductions which this variable bounds manager has detected.

**0.15.586.4.6 getConflict()** template<typename T >  
`std::set<T> smtrat::vb::VariableBounds< T >::getConflict ( ) const [inline]`

##### Returns

The origins which cause the conflict. This method can only be called, if there is a conflict.

**0.15.586.4.7 getDoubleInterval()** template<typename T >  
const carl::Interval<double>& smtrat::vb::VariableBounds< T >::getDoubleInterval (

    const carl::Variable & \_var ) const

Creates a double interval corresponding to the variable bounds of the given variable.

#### Parameters

|      |                                                                     |
|------|---------------------------------------------------------------------|
| _var | The variable to compute the variable bounds as double interval for. |
|------|---------------------------------------------------------------------|

#### Returns

The variable bounds as a double interval.

**0.15.586.4.8 getEvalIntervalMap()** template<typename T >  
const EvalRationalIntervalMap& smtrat::vb::VariableBounds< T >::getEvalIntervalMap ( ) const

Creates an evalintervalmap corresponding to the variable bounds.

#### Returns

The variable bounds as an evalintervalmap.

**0.15.586.4.9 getInterval() [1/3]** template<typename T >  
RationalInterval smtrat::vb::VariableBounds< T >::getInterval (

    carl::Monomial::Arg \_mon ) const

Creates an interval corresponding to the bounds of the given monomial.

#### Parameters

|      |                                                     |
|------|-----------------------------------------------------|
| _mon | The monomial to compute the bounds as interval for. |
|------|-----------------------------------------------------|

#### Returns

The monomial bounds as an interval.

**0.15.586.4.10 getInterval() [2/3]** template<typename T >  
const RationalInterval& smtrat::vb::VariableBounds< T >::getInterval (

    const carl::Variable & \_var ) const

Creates an interval corresponding to the variable bounds of the given variable.

#### Parameters

|      |                                                              |
|------|--------------------------------------------------------------|
| _var | The variable to compute the variable bounds as interval for. |
|------|--------------------------------------------------------------|

#### Returns

The variable bounds as an interval.

**0.15.586.4.11 getInterval() [3/3]** template<typename T >  
RationalInterval smtrat::vb::VariableBounds< T >::getInterval (

    const TermT & \_term ) const

Creates an interval corresponding to the bounds of the given term.

**Parameters**

|                    |                                                 |
|--------------------|-------------------------------------------------|
| <code>_term</code> | The term to compute the bounds as interval for. |
|--------------------|-------------------------------------------------|

**Returns**

The term bounds as an interval.

**0.15.586.4.12 `getIntervalMap()`** template<typename T >  
const `smtrat::EvalDoubleIntervalMap& smtrat::vb::VariableBounds< T >::getIntervalMap ( ) const`  
Creates an interval map corresponding to the variable bounds.

**Returns**

The variable bounds as an interval map.

**0.15.586.4.13 `getOriginSetOfBounds()` [1/3]** template<typename T >  
`std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds ( ) const`

**0.15.586.4.14 `getOriginSetOfBounds()` [2/3]** template<typename T >  
`std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds (`  
`const carl::Variable & _var ) const`

**0.15.586.4.15 `getOriginSetOfBounds()` [3/3]** template<typename T >  
`std::set<T> smtrat::vb::VariableBounds< T >::getOriginSetOfBounds (`  
`const carl::Variables & _variables ) const`

**0.15.586.4.16 `getOriginsOfBounds()` [1/3]** template<typename T >  
`std::vector<T> smtrat::vb::VariableBounds< T >::getOriginsOfBounds ( ) const`  
Collect the origins to the supremums and infimums of all variables.

**Returns**

A set of origins corresponding to the supremums and infimums of all variables.

**0.15.586.4.17 `getOriginsOfBounds()` [2/3]** template<typename T >  
`std::vector<T> smtrat::vb::VariableBounds< T >::getOriginsOfBounds (`  
`const carl::Variable & _var ) const`

**Parameters**

|                   |                                                |
|-------------------|------------------------------------------------|
| <code>_var</code> | The variable to get origins of the bounds for. |
|-------------------|------------------------------------------------|

**Returns**

The origin constraints of the supremum and infimum of the given variable.

**0.15.586.4.18 `getOriginsOfBounds()` [3/3]** template<typename T >

```
std::vector<T> smtrat::vb::VariableBounds< T >::getOriginsOfBounds (
 const carl::Variables & _variables) const
```

**Parameters**

|                         |                                                 |
|-------------------------|-------------------------------------------------|
| <code>_variables</code> | The variables to get origins of the bounds for. |
|-------------------------|-------------------------------------------------|

**Returns**

The origin constraints of the supremum and infimum of the given variables.

**0.15.586.4.19 `isConflicting()`** template<typename T >  
`bool smtrat::vb::VariableBounds< T >::isConflicting ( ) const [inline]`

**Returns**

true, if there is a conflicting variable; false, otherwise.

**0.15.586.4.20 `print()`** template<typename T >  
`void smtrat::vb::VariableBounds< T >::print (
 std::ostream & _out = std::cout,
 const std::string _init = "",
 bool _printAllBounds = false ) const`

Prints the variable bounds.

**Parameters**

|                              |                                                                                  |
|------------------------------|----------------------------------------------------------------------------------|
| <code>_out</code>            | The output stream to print on.                                                   |
| <code>_init</code>           | A string which is displayed at the beginning of every row.                       |
| <code>_printAllBounds</code> | A flag that indicates whether to print also the bounds of each variable (=true). |

**0.15.586.4.21 `removeBound()` [1/3]** template<typename T >  
`unsigned smtrat::vb::VariableBounds< T >::removeBound (
 const ConstraintT & _constraint,
 const T & _origin )`

Removes all effects the given constraint has on the variable bounds.

**Parameters**

|                          |                                                                        |
|--------------------------|------------------------------------------------------------------------|
| <code>_constraint</code> | The constraint, which effects shall be undone for the variable bounds. |
| <code>_origin</code>     | The origin of the given constraint.                                    |

**Returns**

2, if the variables supremum or infimum have been changed; 1, if the constraint was a (not the strictest) bound;  
0, otherwise.

**0.15.586.4.22 `removeBound()` [2/3]** template<typename T >  
`unsigned smtrat::vb::VariableBounds< T >::removeBound (
 const ConstraintT & _constraint,`

```
 const T & _origin,
 carl::Variable *& _changedVariable)
```

Removes all effects the given constraint has on the variable bounds.

#### Parameters

|                               |                                                                                |
|-------------------------------|--------------------------------------------------------------------------------|
| <code>_constraint</code>      | The constraint, which effects shall be undone for the variable bounds.         |
| <code>_origin</code>          | The origin of the given constraint.                                            |
| <code>_changedVariable</code> | The variable whose interval domain has been changed, if the return value is 2. |

#### Returns

2, if the variables supremum or infimum have been changed; 1, if the constraint was a (not the strictest) bound; 0, otherwise.

### 0.15.586.4.23 `removeBound()` [3/3] template<typename T>

```
unsigned smrat::vb::VariableBounds< T >::removeBound (
 const FormulaT & _formula,
 const T & _origin)
```

## 0.15.587 smrat::VariableCapsule Class Reference

```
#include <VariableCapsule.h>
```

#### Public Member Functions

- `VariableCapsule` (const carl::Variable &`xVariable`, const carl::Variable &`yVariable`, const carl::Variable &`zVariable`)
- const carl::Variable & `getXVariable` () const
- const carl::Variable & `getYVariable` () const
- const carl::Variable & `getZVariable` () const

#### Data Fields

- carl::Variable `xVariable`
- carl::Variable `yVariable`
- carl::Variable `zVariable`

### 0.15.587.1 Constructor & Destructor Documentation

```
0.15.587.1.1 VariableCapsule() smrat::VariableCapsule::VariableCapsule (
 const carl::Variable & xVariable,
 const carl::Variable & yVariable,
 const carl::Variable & zVariable)
```

### 0.15.587.2 Member Function Documentation

```
0.15.587.2.1 getXVariable() const carl::Variable & smrat::VariableCapsule::getXVariable ()
```

const

```
0.15.587.2.2 getYVariable() const carl::Variable & smtrat::VariableCapsule::getYVariable ()
const
```

```
0.15.587.2.3 getZVariable() const carl::Variable & smtrat::VariableCapsule::getZVariable ()
const
```

### 0.15.587.3 Field Documentation

```
0.15.587.3.1 xVariable carl::Variable smtrat::VariableCapsule::xVariable
```

```
0.15.587.3.2 yVariable carl::Variable smtrat::VariableCapsule::yVariable
```

```
0.15.587.3.3 zVariable carl::Variable smtrat::VariableCapsule::zVariable
```

## 0.15.588 smtrat::mcsat::variableordering::VariableIDs Struct Reference

```
#include <helper.h>
```

### Public Member Functions

- std::size\_t **operator[]** (carl::Variable v)
- std::size\_t **operator[]** (carl::Variable v) const

### Data Fields

- std::map< carl::Variable, std::size\_t > **mIDs**

### 0.15.588.1 Member Function Documentation

```
0.15.588.1.1 operator[]() [1/2] std::size_t smtrat::mcsat::variableordering::VariableIDs::operator[]
(
 carl::Variable v) [inline]
```

```
0.15.588.1.2 operator[]() [2/2] std::size_t smtrat::mcsat::variableordering::VariableIDs::operator[]
(
 carl::Variable v) const [inline]
```

### 0.15.588.2 Field Documentation

```
0.15.588.2.1 mIDs std::map<carl::Variable, std::size_t> smtrat::mcsat::variableordering::VariableIDs::mIDs
```

## 0.15.589 smtrat::VariableRewriteRule Class Reference

```
#include <VariableRewriteRule.h>
```

## Public Member Functions

- `VariableRewriteRule` (unsigned `varNr`, const `TermT` &`term`, const `carl::BitVector` &`reasons`)
- `~VariableRewriteRule` ()

## Protected Attributes

- unsigned `mVarNr`  
*Rewrite a variable (identified by this number)*
- `TermT mTerm`  
*Rewrite with this term.*
- `carl::BitVector mReasons`  
*Based on this origins.*

### 0.15.589.1 Constructor & Destructor Documentation

#### 0.15.589.1.1 `VariableRewriteRule()` `smtrat::VariableRewriteRule::VariableRewriteRule` (

```
 unsigned varNr,
 const TermT & term,
 const carl::BitVector & reasons) [inline]
```

#### 0.15.589.1.2 `~VariableRewriteRule()` `smtrat::VariableRewriteRule::~VariableRewriteRule` () [inline]

### 0.15.589.2 Field Documentation

#### 0.15.589.2.1 `mReasons` `carl::BitVector smtrat::VariableRewriteRule::mReasons` [protected] Based on this origins.

#### 0.15.589.2.2 `mTerm` `TermT smtrat::VariableRewriteRule::mTerm` [protected] Rewrite with this term.

#### 0.15.589.2.3 `mVarNr` `unsigned smtrat::VariableRewriteRule::mVarNr` [protected] Rewrite a variable (identified by this number)

### 0.15.590 `smtrat::variant_hash_visitor` Struct Reference

```
#include <VariantHash.h>
```

## Public Member Functions

- template<typename T >  
  `std::size_t operator()` (const T &`value`) const noexcept

### 0.15.590.1 Member Function Documentation

#### 0.15.590.1.1 `operator()` template<typename T > `std::size_t smtrat::variant_hash_visitor::operator()` (     const T & `value` ) const [inline], [noexcept]

## 0.15.591 smtrat::parser::conversion::VariantConverter< Res > Struct Template Reference

Converts variants to some type using the [Converter](#) class.

```
#include <Conversions.h>
```

### Public Types

- `typedef bool result_type`

### Public Member Functions

- `template<typename T>  
bool operator()(const T &t)`
- `template<BOOST_VARIANT_ENUM_PARAMS(typename T)>  
bool operator()(const boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)> &t)`
- `template<BOOST_VARIANT_ENUM_PARAMS(typename T)>  
bool operator()(const boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)> &t, Res &r)`
- `template<typename T>  
Res convert(const T &t)`

### Data Fields

- `Res result`
- `Converter< Res > converter`

#### 0.15.591.1 Detailed Description

```
template<typename Res>
struct smtrat::parser::conversion::VariantConverter< Res >
```

Converts variants to some type using the [Converter](#) class.

#### 0.15.591.2 Member Typedef Documentation

**0.15.591.2.1 `result_type`** `template<typename Res >`  
`typedef bool smtrat::parser::conversion::VariantConverter< Res >::result_type`

#### 0.15.591.3 Member Function Documentation

**0.15.591.3.1 `convert()`** `template<typename Res >`  
`template<typename T >`  
`Res smtrat::parser::conversion::VariantConverter< Res >::convert (`  
`const T & t ) [inline]`

**0.15.591.3.2 `operator()()` [1/3]** `template<typename Res >`  
`template<BOOST_VARIANT_ENUM_PARAMS(typename T) >`  
`bool smtrat::parser::conversion::VariantConverter< Res >::operator() (`  
`const boost::variant< BOOST_VARIANT_ENUM_PARAMS(T) > & t ) [inline]`

---

**0.15.591.3.3 operator() [2/3]** template<typename Res >  
 template<BOOST\_VARIANT\_ENUM\_PARAMS(typename T) >  
 bool smtrat::parser::conversion::VariantConverter< Res >::operator() (const boost::variant< BOOST\_VARIANT\_ENUM\_PARAMS(T) > & t, Res & r ) [inline]

**0.15.591.3.4 operator() [3/3]** template<typename Res >  
 template<typename T >  
 bool smtrat::parser::conversion::VariantConverter< Res >::operator() (const T & t ) [inline]

#### 0.15.591.4 Field Documentation

**0.15.591.4.1 converter** template<typename Res >  
 Converter<Res> smtrat::parser::conversion::VariantConverter< Res >::converter

**0.15.591.4.2 result** template<typename Res >  
 Res smtrat::parser::conversion::VariantConverter< Res >::result

### 0.15.592 smtrat::VariantMap< Key, Value > Class Template Reference

This class is a specialization of `std::map` where the keys are of arbitrary type and the values are of type `boost::variant`.

```
#include <VariantMap.h>
```

#### Public Member Functions

- template<typename T , typename Output >  
`void assertType` (const Key &key, Output out) const  
*Asserts that the value that is associated with the given key has a specified type.*
- template<typename T >  
`bool has` (const Key &key) const  
*Checks if there is a value of the specified type for the given key.*
- template<typename T >  
`const T & get` (const Key &key) const  
*Returns the value associated with the given key as type T.*

#### Data Fields

- K keys  
*STL member.*
- T elements  
*STL member.*

#### 0.15.592.1 Detailed Description

```
template<typename Key, typename Value>
class smtrat::VariantMap< Key, Value >
```

This class is a specialization of `std::map` where the keys are of arbitrary type and the values are of type `boost::variant`.

There is no point in using this class if the values are no variants. Most probably, it would not compile anyway and fail to do so with a large bunch of compiler errors. To prevent this, we assert that the value type is indeed a `boost::variant`.

Additionally to the standard methods inherited from `std::map`, it implements three additional methods:

- `assertType<T>(key, out)`: Asserts that there is a value for the given key and that it has the type `T`. If this is not the case, an appropriate error is written to `out`.
- `has<T>(key)`: Checks if there is a value for the given key and if this value has the type `T`.
- `get<T>(key)`: Returns the value for the given key as type `T`.

#### Template Parameters

|                    |                 |
|--------------------|-----------------|
| <code>Key</code>   | Type of keys.   |
| <code>Value</code> | Type of values. |

#### 0.15.592.2 Member Function Documentation

```
0.15.592.2.1 assertType() template<typename Key , typename Value >
template<typename T , typename Output >
void smtrat::VariantMap< Key, Value >::assertType (
 const Key & key,
 Output out) const [inline]
```

Asserts that the value that is associated with the given key has a specified type.

The assertion holds, if

- there is a value in the map for the given key and
- the stored value has the type `T`.

#### Template Parameters

|                     |                                  |
|---------------------|----------------------------------|
| <code>T</code>      | Type that the value should have. |
| <code>Output</code> | Type of out.                     |

#### Parameters

|                  |                                     |
|------------------|-------------------------------------|
| <code>key</code> | Key.                                |
| <code>out</code> | Functor returning an output stream. |

```
0.15.592.2.2 get() template<typename Key , typename Value >
template<typename T >
const T& smtrat::VariantMap< Key, Value >::get (
 const Key & key) const [inline]
```

Returns the value associated with the given key as type `T`.

#### Template Parameters

|                |                    |
|----------------|--------------------|
| <code>T</code> | Type of the value. |
|----------------|--------------------|

**Parameters**

|            |      |
|------------|------|
| <i>key</i> | Key. |
|------------|------|

**Returns**

Value of key as type T.

**0.15.592.2.3 has()** template<typename Key , typename Value >  
template<typename T >

bool smtrat::VariantMap< Key, Value >::has (

    const Key & *key* ) const [inline]

Checks if there is a value of the specified type for the given key.

**Template Parameters**

|          |                                  |
|----------|----------------------------------|
| <i>T</i> | Type that the value should have. |
|----------|----------------------------------|

**Parameters**

|            |      |
|------------|------|
| <i>key</i> | Key. |
|------------|------|

**Returns**

If there is a value of this type.

**0.15.592.3 Field Documentation**

**0.15.592.3.1 elements** T std::map< K, T >::elements [inherited]  
STL member.

**0.15.592.3.2 keys** K std::map< K, T >::keys [inherited]  
STL member.

**0.15.593 smtrat::parser::conversion::VariantVariantConverter< Res > Struct Template Reference**

Converts variants to another variant type not using the [Converter](#) class.

```
#include <Conversions.h>
```

**Public Types**

- **typedef Res result\_type**

**Public Member Functions**

- **template<typename T >**  
    Res [operator\(\)](#) (const T &t)
- **template<typename Variant >**  
    Res [convert](#) (const Variant &t)

### 0.15.593.1 Detailed Description

```
template<typename Res>
struct smtrat::parser::conversion::VariantVariantConverter< Res >
```

Converts variants to another variant type not using the [Converter](#) class.

### 0.15.593.2 Member Typedef Documentation

```
0.15.593.2.1 result_type template<typename Res >
typedef Res smtrat::parser::conversion::VariantVariantConverter< Res >::result_type
```

### 0.15.593.3 Member Function Documentation

```
0.15.593.3.1 convert() template<typename Res >
template<typename Variant >
Res smtrat::parser::conversion::VariantVariantConverter< Res >::convert (
 const Variant & t) [inline]
```

```
0.15.593.3.2 operator() template<typename Res >
template<typename T >
Res smtrat::parser::conversion::VariantVariantConverter< Res >::operator() (
 const T & t) [inline]
```

## 0.15.594 smtrat::VarSchedulerBase Class Reference

Base class for variable schedulers.

```
#include <VarScheduler.h>
```

### Public Member Functions

- template<typename BaseModule >  
`VarSchedulerBase` (BaseModule &baseModule)
- void `rebuild` ()  
*Rebuild heap.*
- void `insert` (Minisat::Var)  
*Insert a variable.*
- Minisat::Lit `pop` ()  
*Returns the next variable to be decided.*
- bool `empty` ()  
*Check if empty.*
- void `print` () const  
*Check if variable is contained.*
- void `increaseActivity` (Minisat::Var)
- void `decreaseActivity` (Minisat::Var)
- void `rebuildActivities` ()
- template<typename Constraints >  
void `rebuildTheoryVars` (const Constraints &)
- void `attachClause` (Minisat::CRef)
- void `detachClause` (Minisat::CRef)
- void `relocateClauses` (std::function< void(Minisat::CRef &) >)

## Protected Attributes

- std::function< double(Minisat::Var)> getActivity
- std::function< char(Minisat::Var)> getPolarity
- std::function< void(Minisat::Var, bool)> setPolarity
- std::function< bool(Minisat::Var)> isDecisionVar
- std::function< bool(Minisat::Var)> isBoolValueUndef
- std::function< bool(Minisat::Var)> isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var)> reabstractVariable
- std::function< const FormulaT &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

### 0.15.594.1 Detailed Description

Base class for variable schedulers.

Removes the need to implement stubs.

During instantiation, the Scheduler is constructed with a const [SATModule](#)& as parameter.

### 0.15.594.2 Constructor & Destructor Documentation

```
0.15.594.2.1 VarSchedulerBase() template<typename BaseModule >
smtrat::VarSchedulerBase::VarSchedulerBase (
 BaseModule & baseModule) [inline]
```

### 0.15.594.3 Member Function Documentation

```
0.15.594.3.1 attachClause() void smtrat::VarSchedulerBase::attachClause (
 Minisat::CRef) [inline]
```

```
0.15.594.3.2 decreaseActivity() void smtrat::VarSchedulerBase::decreaseActivity (
 Minisat::Var) [inline]
```

```
0.15.594.3.3 detachClause() void smtrat::VarSchedulerBase::detachClause (
 Minisat::CRef) [inline]
```

```
0.15.594.3.4 empty() bool smtrat::VarSchedulerBase::empty () [inline]
Check if empty.
```

```
0.15.594.3.5 increaseActivity() void smtrat::VarSchedulerBase::increaseActivity (
 Minisat::Var) [inline]
```

**0.15.594.3.6 `insert()`** void smrat::VarSchedulerBase::insert ( Minisat::Var ) [inline]

Insert a variable.

If already contained, do nothing.

**0.15.594.3.7 `pop()`** Minisat::Lit smrat::VarSchedulerBase::pop ( ) [inline]

Returns the next variable to be decided.

Returns and removes the next variable to be decided.

**0.15.594.3.8 `print()`** void smrat::VarSchedulerBase::print ( ) const [inline]

Check if variable is contained.

Print.

**0.15.594.3.9 `rebuild()`** void smrat::VarSchedulerBase::rebuild ( ) [inline]

Rebuild heap.

**0.15.594.3.10 `rebuildActivities()`** void smrat::VarSchedulerBase::rebuildActivities ( ) [inline]

**0.15.594.3.11 `rebuildTheoryVars()`** template<typename Constraints >  
void smrat::VarSchedulerBase::rebuildTheoryVars ( const Constraints & ) [inline]

**0.15.594.3.12 `relocateClauses()`** void smrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &) > ) [inline]

## 0.15.594.4 Field Documentation

**0.15.594.4.1 `abstractLiteral`** std::function<const Minisat::Lit (const FormulaT&)> smrat::VarSchedulerBase::abstractLiteral [protected]

**0.15.594.4.2 `abstractVariable`** std::function< Minisat::Var (const FormulaT&)> smrat::VarSchedulerBase::abstractVariable [protected]

**0.15.594.4.3 `currentlySatisfiedByBackend`** std::function<unsigned(const FormulaT&)> smrat::VarSchedulerBase::currentlySatisfiedByBackend [protected]

**0.15.594.4.4 `getActivity`** std::function<double(Minisat::Var)> smrat::VarSchedulerBase::getActivity [protected]

**0.15.594.4.5 `getBoolLitValue`** std::function<Minisat::lbool(Minisat::Lit)> smrat::VarSchedulerBase::getBoolLitValue [protected]

**0.15.594.4.6 `getBoolVarValue`** std::function<Minisat::lbool(Minisat::Var)> smrat::VarSchedulerBase::getBoolVarValue [protected]

**0.15.594.4.7 `getClause`** std::function<const Minisat::Clause& (Minisat::CRef)> smrat::VarSchedulerBase::getClause [protected]

**0.15.594.4.8 `getPolarity`** std::function<char (Minisat::Var)> smrat::VarSchedulerBase::getPolarity [protected]

**0.15.594.4.9 `isAbstractedFormula`** std::function<bool (const FormulaT&)> smrat::VarSchedulerBase::isAbstractedFormula [protected]

**0.15.594.4.10 `isBoolValueUndef`** std::function<bool (Minisat::Var)> smrat::VarSchedulerBase::isBoolValueUndef [protected]

**0.15.594.4.11 `isDecisionVar`** std::function<bool (Minisat::Var)> smrat::VarSchedulerBase::isDecisionVar [protected]

**0.15.594.4.12 `isTheoryAbstraction`** std::function<bool (Minisat::Var)> smrat::VarSchedulerBase::isTheoryAbstraction [protected]

**0.15.594.4.13 `reabstractLiteral`** std::function<const FormulaT& (Minisat::Lit)> smrat::VarSchedulerBase::reabstractLiteral [protected]

**0.15.594.4.14 `reabstractVariable`** std::function<const FormulaT& (Minisat::Var)> smrat::VarSchedulerBase::reabstractVariable [protected]

**0.15.594.4.15 `setPolarity`** std::function<void (Minisat::Var, bool)> smrat::VarSchedulerBase::setPolarity [protected]

## 0.15.595 smrat::VarSchedulerFixedRandom Class Reference

```
#include <VarScheduler.h>
```

### Public Member Functions

- template<typename BaseModule>  
  VarSchedulerFixedRandom (BaseModule &baseModule)
- void `rebuild` ()
- void `insert` (Minisat::Var var)
- Minisat::Var `top` ()
- Minisat::Lit `pop` ()
- bool `empty` ()
- void `print` () const
- void `increaseActivity` (Minisat::Var var)
- void `decreaseActivity` (Minisat::Var var)
- void `rebuildActivities` ()
- template<typename Constraints>  
  void `rebuildTheoryVars` (const Constraints &)
- void `attachClause` (Minisat::CRef)
- void `detachClause` (Minisat::CRef)
- void `relocateClauses` (std::function<void(Minisat::CRef &)>)

**Protected Member Functions**

- bool `valid (Minisat::Var var)`

**Protected Attributes**

- std::function< double(Minisat::Var)> `getActivity`
- std::function< char(Minisat::Var)> `getPolarity`
- std::function< void(Minisat::Var, bool)> `setPolarity`
- std::function< bool(Minisat::Var)> `isDecisionVar`
- std::function< bool(Minisat::Var)> `isBoolValueUndef`
- std::function< bool(Minisat::Var)> `isTheoryAbstraction`
- std::function< const FormulaT &(Minisat::Var)> `reabstractVariable`
- std::function< const FormulaT &(Minisat::Lit)> `reabstractLiteral`
- std::function< const Minisat::Clause &(Minisat::CRef)> `getClause`
- std::function< Minisat::lbool(Minisat::Var)> `getBoolVarValue`
- std::function< Minisat::lbool(Minisat::Lit)> `getBoolLitValue`
- std::function< unsigned(const FormulaT &)> `currentlySatisfiedByBackend`
- std::function< Minisat::Var(const FormulaT &)> `abstractVariable`
- std::function< const Minisat::Lit(const FormulaT &)> `abstractLiteral`
- std::function< bool(const FormulaT &)> `isAbstractedFormula`

**0.15.595.1 Constructor & Destructor Documentation**

**0.15.595.1.1 VarSchedulerFixedRandom()** template<typename BaseModule >  
`smtrat:::VarSchedulerFixedRandom:::VarSchedulerFixedRandom (`  
`BaseModule & baseModule ) [inline], [explicit]`

**0.15.595.2 Member Function Documentation**

**0.15.595.2.1 attachClause()** void `smtrat:::VarSchedulerBase:::attachClause (`  
`Minisat:::CRef ) [inline], [inherited]`

**0.15.595.2.2 decreaseActivity()** void `smtrat:::VarSchedulerMinisat:::decreaseActivity (`  
`Minisat:::Var var ) [inline], [inherited]`

**0.15.595.2.3 detachClause()** void `smtrat:::VarSchedulerBase:::detachClause (`  
`Minisat:::CRef ) [inline], [inherited]`

**0.15.595.2.4 empty()** bool `smtrat:::VarSchedulerMinisat:::empty ( ) [inline], [inherited]`

**0.15.595.2.5 increaseActivity()** void `smtrat:::VarSchedulerMinisat:::increaseActivity (`  
`Minisat:::Var var ) [inline], [inherited]`

**0.15.595.2.6 insert()** void `smtrat:::VarSchedulerMinisat:::insert (`  
`Minisat:::Var var ) [inline], [inherited]`

**0.15.595.2.7 `pop()`** `Minisat::Lit` `smtrat::VarSchedulerMinisat::pop()` [inline], [inherited]

**0.15.595.2.8 `print()`** `void smtrat::VarSchedulerMinisat::print()` const [inline], [inherited]

**0.15.595.2.9 `rebuild()`** `void smtrat::VarSchedulerMinisat::rebuild()` [inline], [inherited]

**0.15.595.2.10 `rebuildActivities()`** `void smtrat::VarSchedulerMinisat::rebuildActivities()` [inline], [inherited]

**0.15.595.2.11 `rebuildTheoryVars()`** template<typename Constraints>  
`void smtrat::VarSchedulerBase::rebuildTheoryVars(`  
    `const Constraints & )` [inline], [inherited]

**0.15.595.2.12 `relocateClauses()`** `void smtrat::VarSchedulerBase::relocateClauses(`  
    `std::function< void(Minisat::CRef &) > )` [inline], [inherited]

**0.15.595.2.13 `top()`** `Minisat::Var` `smtrat::VarSchedulerMinisat::top()` [inline], [inherited]

**0.15.595.2.14 `valid()`** `bool smtrat::VarSchedulerMinisat::valid(`  
    `Minisat::Var var)` [inline], [protected], [inherited]

### 0.15.595.3 Field Documentation

**0.15.595.3.1 `abstractLiteral`** `std::function<const Minisat::Lit(const FormulaT&)>` `smtrat::VarSchedulerBase::abstractLiteral` [protected], [inherited]

**0.15.595.3.2 `abstractVariable`** `std::function< Minisat::Var(const FormulaT&)>` `smtrat::VarSchedulerBase::abstractVariable` [protected], [inherited]

**0.15.595.3.3 `currentlySatisfiedByBackend`** `std::function<unsigned(const FormulaT&)>` `smtrat::VarSchedulerBase::currentlySatisfiedByBackend` [protected], [inherited]

**0.15.595.3.4 `getActivity`** `std::function<double(Minisat::Var)>` `smtrat::VarSchedulerBase::getActivity` [protected], [inherited]

**0.15.595.3.5 `getBoolLitValue`** `std::function<Minisat::lbool(Minisat::Lit)>` `smtrat::VarSchedulerBase::getBoolLitValue` [protected], [inherited]

**0.15.595.3.6 `getBoolVarValue`** `std::function<Minisat::lbool(Minisat::Var)>` `smtrat::VarSchedulerBase::getBoolVarValue` [protected], [inherited]

**0.15.595.3.7 `getClause`** std::function<const `Minisat::Clause`& (`Minisat::CRef`)> `smtrat::Var`↔  
`SchedulerBase::getClause` [protected], [inherited]

**0.15.595.3.8 `getPolarity`** std::function<char (`Minisat::Var`)> `smtrat::VarSchedulerBase::get`↔  
`Polarity` [protected], [inherited]

**0.15.595.3.9 `isAbstractedFormula`** std::function<bool (const `FormulaT`&)> `smtrat::VarScheduler`↔  
`Base::isAbstractedFormula` [protected], [inherited]

**0.15.595.3.10 `isBoolValueUndef`** std::function<bool (`Minisat::Var`)> `smtrat::VarSchedulerBase::is`↔  
`::isBoolValueUndef` [protected], [inherited]

**0.15.595.3.11 `isDecisionVar`** std::function<bool (`Minisat::Var`)> `smtrat::VarSchedulerBase::is`↔  
`DecisionVar` [protected], [inherited]

**0.15.595.3.12 `isTheoryAbstraction`** std::function<bool (`Minisat::Var`)> `smtrat::VarSchedulerBase::is`↔  
`::isTheoryAbstraction` [protected], [inherited]

**0.15.595.3.13 `reabstractLiteral`** std::function<const `FormulaT`& (`Minisat::Lit`)> `smtrat::VarScheduler`↔  
`Base::reabstractLiteral` [protected], [inherited]

**0.15.595.3.14 `reabstractVariable`** std::function<const `FormulaT`& (`Minisat::Var`)> `smtrat::Var`↔  
`SchedulerBase::reabstractVariable` [protected], [inherited]

**0.15.595.3.15 `setPolarity`** std::function<void (`Minisat::Var`, bool)> `smtrat::VarSchedulerBase::set`↔  
`::setPolarity` [protected], [inherited]

## 0.15.596 `smtrat::VarSchedulerMcsatActivityPreferTheory`< `vot` > Class Template Reference

Activity-based scheduler preferring initially theory variables.

```
#include <VarSchedulerMcsat.h>
```

### Public Member Functions

- template<typename BaseModule >  
`VarSchedulerMcsatActivityPreferTheory` (`BaseModule &baseModule`)
- void `rebuild` ()
- void `insert` (`Minisat::Var` var)
- `Minisat::Lit` `pop` ()
- bool `empty` ()
- void `print` () const
- void `increaseActivity` (`Minisat::Var` var)
- void `decreaseActivity` (`Minisat::Var` var)
- void `rebuildActivities` ()
- template<typename Constraints >  
`void` `rebuildTheoryVars` (const `Constraints &c`)

- void `attachClause` (`Minisat::CRef`)
- void `detachClause` (`Minisat::CRef`)
- void `relocateClauses` (`std::function< void(Minisat::CRef &) >`)

### Protected Attributes

- `std::function< bool(Minisat::Var) >` `isTheoryVar`
- `std::function< carl::Variable(Minisat::Var) >` `carlVar`
- `std::function< Minisat::Var(carl::Variable) >` `minisatVar`
- `std::function< Model() >` `currentModel`
- `std::function< double(Minisat::Var) >` `getActivity`
- `std::function< char(Minisat::Var) >` `getPolarity`
- `std::function< void(Minisat::Var, bool) >` `setPolarity`
- `std::function< bool(Minisat::Var) >` `isDecisionVar`
- `std::function< bool(Minisat::Var) >` `isBoolValueUndef`
- `std::function< bool(Minisat::Var) >` `isTheoryAbstraction`
- `std::function< const FormulaT &(Minisat::Var) >` `reabstractVariable`
- `std::function< const FormulaT &(Minisat::Lit) >` `reabstractLiteral`
- `std::function< const Minisat::Clause &(Minisat::CRef) >` `getClause`
- `std::function< Minisat::lbool(Minisat::Var) >` `getBoolVarValue`
- `std::function< Minisat::lbool(Minisat::Lit) >` `getBoolLitValue`
- `std::function< unsigned(const FormulaT &) >` `currentlySatisfiedByBackend`
- `std::function< Minisat::Var(const FormulaT &) >` `abstractVariable`
- `std::function< const Minisat::Lit(const FormulaT &) >` `abstractLiteral`
- `std::function< bool(const FormulaT &) >` `isAbstractedFormula`

#### 0.15.596.1 Detailed Description

```
template<mcsat::VariableOrdering vot>
class smrat::VarSchedulerMcsatActivityPreferTheory< vot >
```

Activity-based scheduler preferring initially theory variables.

#### 0.15.596.2 Constructor & Destructor Documentation

```
0.15.596.2.1 VarSchedulerMcsatActivityPreferTheory() template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smrat::VarSchedulerMcsatActivityPreferTheory< vot >::VarSchedulerMcsatActivityPreferTheory (
 BaseModule & baseModule) [inline], [explicit]
```

#### 0.15.596.3 Member Function Documentation

```
0.15.596.3.1 attachClause() void smrat::VarSchedulerBase::attachClause (
 Minisat::CRef) [inline], [inherited]
```

```
0.15.596.3.2 decreaseActivity() template<mcsat::VariableOrdering vot>
void smrat::VarSchedulerMcsatActivityPreferTheory< vot >::decreaseActivity (
 Minisat::Var var) [inline]
```

```
0.15.596.3.3 detachClause() void smrat::VarSchedulerBase::detachClause (
 Minisat::CRef) [inline], [inherited]
```

**0.15.596.3.4 empty()** template<mcsat::VariableOrdering vot>  
bool smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::empty ( ) [inline]

**0.15.596.3.5 increaseActivity()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::increaseActivity ( Minisat::Var var ) [inline]

**0.15.596.3.6 insert()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::insert ( Minisat::Var var ) [inline]

**0.15.596.3.7 pop()** template<mcsat::VariableOrdering vot>  
Minisat::Lit smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::pop ( ) [inline]

**0.15.596.3.8 print()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::print ( ) const [inline]

**0.15.596.3.9 rebuild()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuild ( ) [inline]

**0.15.596.3.10 rebuildActivities()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuildActivities ( ) [inline]

**0.15.596.3.11 rebuildTheoryVars()** template<mcsat::VariableOrdering vot>  
template<typename Constraints >  
void smtrat::VarSchedulerMcsatActivityPreferTheory< vot >::rebuildTheoryVars ( const Constraints & c ) [inline]

**0.15.596.3.12 relocateClauses()** void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &) > ) [inline], [inherited]

## 0.15.596.4 Field Documentation

**0.15.596.4.1 abstractLiteral** std::function<const Minisat::Lit (const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]

**0.15.596.4.2 abstractVariable** std::function< Minisat::Var (const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]

**0.15.596.4.3 carlVar** std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase::carlVar [protected], [inherited]

**0.15.596.4.4 currentlySatisfiedByBackend** std::function<unsigned(const [FormulaT](#)&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.596.4.5 currentModel** std::function<[Model](#)()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]

**0.15.596.4.6 getActivity** std::function<double([Minisat::Var](#))> smtrat::VarSchedulerBase::getActivity [protected], [inherited]

**0.15.596.4.7 getBoolLitValue** std::function<[Minisat::lbool](#)([Minisat::Lit](#))> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

**0.15.596.4.8 getBoolVarValue** std::function<[Minisat::lbool](#)([Minisat::Var](#))> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

**0.15.596.4.9 getClause** std::function<const [Minisat::Clause](#)&([Minisat::CRef](#))> smtrat::VarSchedulerBase::getClause [protected], [inherited]

**0.15.596.4.10 getPolarity** std::function<char([Minisat::Var](#))> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

**0.15.596.4.11 isAbstractedFormula** std::function<bool(const [FormulaT](#)&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

**0.15.596.4.12 isBoolValueUndef** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]

**0.15.596.4.13 isDecisionVar** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

**0.15.596.4.14 isTheoryAbstraction** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]

**0.15.596.4.15 isTheoryVar** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]

**0.15.596.4.16 minisatVar** std::function<[Minisat::Var](#)(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]

**0.15.596.4.17 reabstractLiteral** std::function<const [FormulaT](#)&([Minisat::Lit](#))> smtrat::VarSchedulerBase::reabstractLiteral [protected], [inherited]

---

**0.15.596.4.18 reabstractVariable** std::function<const [FormulaT](#)&(Minisat::Var)> smtrat::Var<~  
SchedulerBase::reabstractVariable [protected], [inherited]

**0.15.596.4.19 setPolarity** std::function<void(Minisat::Var, bool)> smtrat::VarSchedulerBase<~  
::setPolarity [protected], [inherited]

## 0.15.597 smtrat::VarSchedulerMcsatBase Class Reference

Base class for all MCSAT variable scheduler.

```
#include <VarSchedulerMcsat.h>
```

### Public Member Functions

- template<typename BaseModule >  
[VarSchedulerMcsatBase](#) (BaseModule &baseModule)
- void [rebuild](#) ()  
*Rebuild heap.*
- void [insert](#) (Minisat::Var)  
*Insert a variable.*
- Minisat::Lit [pop](#) ()  
*Returns the next variable to be decided.*
- bool [empty](#) ()  
*Check if empty.*
- void [print](#) () const  
*Check if variable is contained.*
- void [increaseActivity](#) (Minisat::Var)
- void [decreaseActivity](#) (Minisat::Var)
- void [rebuildActivities](#) ()
- template<typename Constraints >  
void [rebuildTheoryVars](#) (const Constraints &)
- void [attachClause](#) (Minisat::CRef)
- void [detachClause](#) (Minisat::CRef)
- void [relocateClauses](#) (std::function< void(Minisat::CRef &)>)

### Protected Attributes

- std::function< bool(Minisat::Var)> [isTheoryVar](#)
- std::function< carl::Variable(Minisat::Var)> [carlVar](#)
- std::function< Minisat::Var(carl::Variable)> [minisatVar](#)
- std::function< Model()> [currentModel](#)
- std::function< double(Minisat::Var)> [getActivity](#)
- std::function< char(Minisat::Var)> [getPolarity](#)
- std::function< void(Minisat::Var, bool)> [setPolarity](#)
- std::function< bool(Minisat::Var)> [isDecisionVar](#)
- std::function< bool(Minisat::Var)> [isBoolValueUndef](#)
- std::function< bool(Minisat::Var)> [isTheoryAbstraction](#)
- std::function< const [FormulaT](#) &(Minisat::Var)> [reabstractVariable](#)
- std::function< const [FormulaT](#) &(Minisat::Lit)> [reabstractLiteral](#)
- std::function< const Minisat::Clause &(Minisat::CRef)> [getClause](#)
- std::function< Minisat::Ibool(Minisat::Var)> [getBoolVarValue](#)
- std::function< Minisat::Ibool(Minisat::Lit)> [getBoolLitValue](#)
- std::function< unsigned(const [FormulaT](#) &)> [currentlySatisfiedByBackend](#)
- std::function< Minisat::Var(const [FormulaT](#) &)> [abstractVariable](#)
- std::function< const Minisat::Lit(const [FormulaT](#) &)> [abstractLiteral](#)
- std::function< bool(const [FormulaT](#) &)> [isAbstractedFormula](#)

### 0.15.597.1 Detailed Description

Base class for all MCSAT variable scheduler.  
Should not be used directly.

### 0.15.597.2 Constructor & Destructor Documentation

**0.15.597.2.1 VarSchedulerMcsatBase()** template<typename BaseModule >  
smtrat::VarSchedulerMcsatBase::VarSchedulerMcsatBase (

```
 BaseModule & baseModule) [inline]
```

### 0.15.597.3 Member Function Documentation

**0.15.597.3.1 attachClause()** void smtrat::VarSchedulerBase::attachClause (

```
 Minisat::CRef) [inline], [inherited]
```

**0.15.597.3.2 decreaseActivity()** void smtrat::VarSchedulerBase::decreaseActivity (

```
 Minisat::Var) [inline], [inherited]
```

**0.15.597.3.3 detachClause()** void smtrat::VarSchedulerBase::detachClause (

```
 Minisat::CRef) [inline], [inherited]
```

**0.15.597.3.4 empty()** bool smtrat::VarSchedulerBase::empty ( ) [inline], [inherited]  
Check if empty.

**0.15.597.3.5 increaseActivity()** void smtrat::VarSchedulerBase::increaseActivity (

```
 Minisat::Var) [inline], [inherited]
```

**0.15.597.3.6 insert()** void smtrat::VarSchedulerBase::insert (

```
 Minisat::Var) [inline], [inherited]
```

Insert a variable.  
If already contained, do nothing.

**0.15.597.3.7 pop()** Minisat::Lit smtrat::VarSchedulerBase::pop ( ) [inline], [inherited]  
Returns the next variable to be decided.  
Returns and removes the next variable to be decided.

**0.15.597.3.8 print()** void smtrat::VarSchedulerBase::print ( ) const [inline], [inherited]  
Check if variable is contained.  
Print.

**0.15.597.3.9 rebuild()** void smtrat::VarSchedulerBase::rebuild ( ) [inline], [inherited]  
Rebuild heap.

**0.15.597.3.10 rebuildActivities()** void smtrat::VarSchedulerBase::rebuildActivities ( ) [inline], [inherited]

**0.15.597.3.11 rebuildTheoryVars()** template<typename Constraints >  
void smtrat::VarSchedulerBase::rebuildTheoryVars (  
    const Constraints & ) [inline], [inherited]

**0.15.597.3.12 relocateClauses()** void smtrat::VarSchedulerBase::relocateClauses (  
    std::function< void(Minisat::CRef &) > ) [inline], [inherited]

#### 0.15.597.4 Field Documentation

**0.15.597.4.1 abstractLiteral** std::function<const Minisat::Lit (const FormulaT&) > smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]

**0.15.597.4.2 abstractVariable** std::function< Minisat::Var (const FormulaT&) > smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]

**0.15.597.4.3 carlVar** std::function<carl::Variable(Minisat::Var) > smtrat::VarSchedulerMcsatBase::carlVar [protected]

**0.15.597.4.4 currentlySatisfiedByBackend** std::function<unsigned(const FormulaT&) > smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.597.4.5 currentModel** std::function<Model() > smtrat::VarSchedulerMcsatBase::currentModel [protected]

**0.15.597.4.6 getActivity** std::function<double(Minisat::Var) > smtrat::VarSchedulerBase::getActivity [protected], [inherited]

**0.15.597.4.7 getBoolLitValue** std::function<Minisat::lbool(Minisat::Lit) > smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

**0.15.597.4.8 getBoolVarValue** std::function<Minisat::lbool(Minisat::Var) > smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

**0.15.597.4.9 getClause** std::function<const Minisat::Clause& (Minisat::CRef) > smtrat::VarSchedulerBase::getClause [protected], [inherited]

**0.15.597.4.10 getPolarity** std::function<char(Minisat::Var) > smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

**0.15.597.4.11 isAbstractedFormula** std::function<bool(const FormulaT&) > smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

**0.15.597.4.12 isBoolValueUndef** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]

**0.15.597.4.13 isDecisionVar** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

**0.15.597.4.14 isTheoryAbstraction** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]

**0.15.597.4.15 isTheoryVar** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected]

**0.15.597.4.16 minisatVar** std::function<[Minisat::Var](#)(carl::Variable)> smtrat::VarScheduler::minisatVar [protected]

**0.15.597.4.17 reabstractLiteral** std::function<const [FormulaT](#)& ([Minisat::Lit](#))> smtrat::VarScheduler::reabstractLiteral [protected], [inherited]

**0.15.597.4.18 reabstractVariable** std::function<const [FormulaT](#)& ([Minisat::Var](#))> smtrat::VarScheduler::reabstractVariable [protected], [inherited]

**0.15.597.4.19 setPolarity** std::function<void([Minisat::Var](#), bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]

## 0.15.598 smtrat::VarSchedulerMcsatBooleanFirst< vot > Class Template Reference

Variable scheduling that all decides Boolean variables first before deciding any theory variable.

```
#include <VarSchedulerMcsat.h>
```

### Public Member Functions

- template<typename BaseModule >  
  [VarSchedulerMcsatBooleanFirst](#) (BaseModule &baseModule)
- void [rebuild](#) ()
- void [insert](#) ([Minisat::Var](#) var)
- [Minisat::Lit](#) [pop](#) ()
- bool [empty](#) ()
- void [print](#) () const
- void [increaseActivity](#) ([Minisat::Var](#) var)
- void [decreaseActivity](#) ([Minisat::Var](#) var)
- void [rebuildActivities](#) ()
- template<typename Constraints >  
  void [rebuildTheoryVars](#) (const Constraints &c)
- void [attachClause](#) ([Minisat::CRef](#))
- void [detachClause](#) ([Minisat::CRef](#))
- void [relocateClauses](#) (std::function< void([Minisat::CRef](#) &) >)

## Protected Attributes

- std::function< bool(Minisat::Var) > `isTheoryVar`
- std::function< carl::Variable(Minisat::Var) > `carlVar`
- std::function< Minisat::Var(carl::Variable) > `minisatVar`
- std::function< Model() > `currentModel`
- std::function< double(Minisat::Var) > `getActivity`
- std::function< char(Minisat::Var) > `getPolarity`
- std::function< void(Minisat::Var, bool) > `setPolarity`
- std::function< bool(Minisat::Var) > `isDecisionVar`
- std::function< bool(Minisat::Var) > `isBoolValueUndef`
- std::function< bool(Minisat::Var) > `isTheoryAbstraction`
- std::function< const FormulaT &(Minisat::Var) > `reabstractVariable`
- std::function< const FormulaT &(Minisat::Lit) > `reabstractLiteral`
- std::function< const Minisat::Clause &(Minisat::CRef) > `getClause`
- std::function< Minisat::lbool(Minisat::Var) > `getBoolVarValue`
- std::function< Minisat::lbool(Minisat::Lit) > `getBoolLitValue`
- std::function< unsigned(const FormulaT &) > `currentlySatisfiedByBackend`
- std::function< Minisat::Var(const FormulaT &) > `abstractVariable`
- std::function< const Minisat::Lit(const FormulaT &) > `abstractLiteral`
- std::function< bool(const FormulaT &) > `isAbstractedFormula`

### 0.15.598.1 Detailed Description

```
template<mcsat::VariableOrdering vot>
class smrat::VarSchedulerMcsatBooleanFirst< vot >
```

Variable scheduling that all decides Boolean variables first before deciding any theory variable.

### 0.15.598.2 Constructor & Destructor Documentation

```
0.15.598.2.1 VarSchedulerMcsatBooleanFirst() template<mcsat::VariableOrdering vot>
template<typename BaseModule >
smrat::VarSchedulerMcsatBooleanFirst< vot >::VarSchedulerMcsatBooleanFirst (
 BaseModule & baseModule) [inline], [explicit]
```

### 0.15.598.3 Member Function Documentation

```
0.15.598.3.1 attachClause() void smrat::VarSchedulerBase::attachClause (
 Minisat::CRef) [inline], [inherited]
```

```
0.15.598.3.2 decreaseActivity() template<mcsat::VariableOrdering vot>
void smrat::VarSchedulerMcsatBooleanFirst< vot >::decreaseActivity (
 Minisat::Var var) [inline]
```

```
0.15.598.3.3 detachClause() void smrat::VarSchedulerBase::detachClause (
 Minisat::CRef) [inline], [inherited]
```

```
0.15.598.3.4 empty() template<mcsat::VariableOrdering vot>
bool smrat::VarSchedulerMcsatBooleanFirst< vot >::empty () [inline]
```

**0.15.598.3.5 increaseActivity()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::increaseActivity ( Minisat::Var var ) [inline]

**0.15.598.3.6 insert()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::insert ( Minisat::Var var ) [inline]

**0.15.598.3.7 pop()** template<mcsat::VariableOrdering vot>  
Minisat::Lit smtrat::VarSchedulerMcsatBooleanFirst< vot >::pop ( ) [inline]

**0.15.598.3.8 print()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::print ( ) const [inline]

**0.15.598.3.9 rebuild()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuild ( ) [inline]

**0.15.598.3.10 rebuildActivities()** template<mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuildActivities ( ) [inline]

**0.15.598.3.11 rebuildTheoryVars()** template<mcsat::VariableOrdering vot>  
template<typename Constraints >  
void smtrat::VarSchedulerMcsatBooleanFirst< vot >::rebuildTheoryVars ( const Constraints & c ) [inline]

**0.15.598.3.12 relocateClauses()** void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &) > ) [inline], [inherited]

## 0.15.598.4 Field Documentation

**0.15.598.4.1 abstractLiteral** std::function<const Minisat::Lit (const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]

**0.15.598.4.2 abstractVariable** std::function< Minisat::Var (const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]

**0.15.598.4.3 carlVar** std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase::carlVar [protected], [inherited]

**0.15.598.4.4 currentlySatisfiedByBackend** std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.598.4.5 `currentModel`** std::function<[Model](#)()> smtrat::VarSchedulerMcsatBase::currentModel  
[protected], [inherited]

**0.15.598.4.6 `getActivity`** std::function<double([Minisat::Var](#))> smtrat::VarSchedulerBase::getActivity  
[protected], [inherited]

**0.15.598.4.7 `getBoolLitValue`** std::function<[Minisat::lbool](#)([Minisat::Lit](#))> smtrat::VarSchedulerBase::getBoolLitValue  
[protected], [inherited]

**0.15.598.4.8 `getBoolVarValue`** std::function<[Minisat::lbool](#)([Minisat::Var](#))> smtrat::VarSchedulerBase::getBoolVarValue  
[protected], [inherited]

**0.15.598.4.9 `getClause`** std::function<const [Minisat::Clause](#)&([Minisat::CRef](#))> smtrat::VarSchedulerBase::getClause  
[protected], [inherited]

**0.15.598.4.10 `getPolarity`** std::function<char([Minisat::Var](#))> smtrat::VarSchedulerBase::getPolarity  
[protected], [inherited]

**0.15.598.4.11 `isAbstractedFormula`** std::function<bool(const [FormulaT](#)&)> smtrat::VarSchedulerBase::isAbstractedFormula  
[protected], [inherited]

**0.15.598.4.12 `isBoolValueUndef`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isBoolValueUndef  
[protected], [inherited]

**0.15.598.4.13 `isDecisionVar`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isDecisionVar  
[protected], [inherited]

**0.15.598.4.14 `isTheoryAbstraction`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isTheoryAbstraction  
[protected], [inherited]

**0.15.598.4.15 `isTheoryVar`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerMcsatBase::isTheoryVar  
[protected], [inherited]

**0.15.598.4.16 `minisatVar`** std::function<[Minisat::Var](#)([carl::Variable](#))> smtrat::VarSchedulerMcsatBase::minisatVar  
[protected], [inherited]

**0.15.598.4.17 `reabstractionLiteral`** std::function<const [FormulaT](#)&([Minisat::Lit](#))> smtrat::VarSchedulerBase::reabstractionLiteral  
[protected], [inherited]

**0.15.598.4.18 `reabstractionVariable`** std::function<const [FormulaT](#)&([Minisat::Var](#))> smtrat::VarSchedulerBase::reabstractionVariable  
[protected], [inherited]

```
0.15.598.4.19 setPolarity std::function<void(Minisat::Var,bool)> smtrat::VarSchedulerBase::
::setPolarity [protected], [inherited]
```

## 0.15.599 smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler > Class Template Reference

Variable scheduling that all decides theory variables first before deciding any Boolean variable.

```
#include <VarSchedulerMcsat.h>
```

### Public Member Functions

- template<typename BaseModule>  
[VarSchedulerMcsatTheoryFirst](#)(BaseModule &baseModule)
- void [rebuild](#)()
- void [insert](#)([Minisat::Var](#) var)
- [Minisat::Lit](#) [pop](#)()
- bool [empty](#)()
- void [print](#)() const
- void [increaseActivity](#)([Minisat::Var](#) var)
- void [decreaseActivity](#)([Minisat::Var](#) var)
- void [rebuildActivities](#)()
- template<typename Constraints>  
void [rebuildTheoryVars](#)(const Constraints &c)
- void [attachClause](#)([Minisat::CRef](#))
- void [detachClause](#)([Minisat::CRef](#))
- void [relocateClauses](#)(std::function<void([Minisat::CRef](#) &)>)

### Protected Attributes

- std::function<bool([Minisat::Var](#))> [isTheoryVar](#)
- std::function<carl::Variable([Minisat::Var](#))> [carlVar](#)
- std::function<[Minisat::Var](#)(carl::Variable)> [minisatVar](#)
- std::function<Model()> [currentModel](#)
- std::function<double([Minisat::Var](#))> [getActivity](#)
- std::function<char([Minisat::Var](#))> [getPolarity](#)
- std::function<void([Minisat::Var](#), bool)> [setPolarity](#)
- std::function<bool([Minisat::Var](#))> [isDecisionVar](#)
- std::function<bool([Minisat::Var](#))> [isBoolValueUndef](#)
- std::function<bool([Minisat::Var](#))> [isTheoryAbstraction](#)
- std::function<const [FormulaT](#) &([Minisat::Var](#))> [reabstractVariable](#)
- std::function<const [FormulaT](#) &([Minisat::Lit](#))> [reabstractLiteral](#)
- std::function<const [Minisat::Clause](#) &([Minisat::CRef](#))> [getClause](#)
- std::function<[Minisat::lbool](#)([Minisat::Var](#))> [getBoolVarValue](#)
- std::function<[Minisat::lbool](#)([Minisat::Lit](#))> [getBoolLitValue](#)
- std::function<unsigned(const [FormulaT](#) &) > [currentlySatisfiedByBackend](#)
- std::function<[Minisat::Var](#)(const [FormulaT](#) &)> [abstractVariable](#)
- std::function<const [Minisat::Lit](#)(const [FormulaT](#) &)> [abstractLiteral](#)
- std::function<bool(const [FormulaT](#) &)> [isAbstractedFormula](#)

### 0.15.599.1 Detailed Description

```
template<typename TheoryScheduler>
class smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >
```

Variable scheduling that all decides theory variables first before deciding any Boolean variable.

## 0.15.599.2 Constructor & Destructor Documentation

**0.15.599.2.1 VarSchedulerMcsatTheoryFirst()** template<typename TheoryScheduler >  
template<typename BaseModule >  
smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::VarSchedulerMcsatTheoryFirst (   
    BaseModule & baseModule ) [inline], [explicit]

## 0.15.599.3 Member Function Documentation

**0.15.599.3.1 attachClause()** void smtrat::VarSchedulerBase::attachClause (   
    Minisat::CRef ) [inline], [inherited]

**0.15.599.3.2 decreaseActivity()** template<typename TheoryScheduler >  
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::decreaseActivity (   
    Minisat::Var var ) [inline]

**0.15.599.3.3 detachClause()** void smtrat::VarSchedulerBase::detachClause (   
    Minisat::CRef ) [inline], [inherited]

**0.15.599.3.4 empty()** template<typename TheoryScheduler >  
bool smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::empty () [inline]

**0.15.599.3.5 increaseActivity()** template<typename TheoryScheduler >  
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::increaseActivity (   
    Minisat::Var var ) [inline]

**0.15.599.3.6 insert()** template<typename TheoryScheduler >  
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::insert (   
    Minisat::Var var ) [inline]

**0.15.599.3.7 pop()** template<typename TheoryScheduler >  
Minisat::Lit smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::pop () [inline]

**0.15.599.3.8 print()** template<typename TheoryScheduler >  
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::print () const [inline]

**0.15.599.3.9 rebuild()** template<typename TheoryScheduler >  
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuild () [inline]

**0.15.599.3.10 rebuildActivities()** template<typename TheoryScheduler >  
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuildActivities () [inline]

```
0.15.599.3.11 rebuildTheoryVars() template<typename TheoryScheduler >
template<typename Constraints >
void smtrat::VarSchedulerMcsatTheoryFirst< TheoryScheduler >::rebuildTheoryVars (
 const Constraints & c) [inline]
```

```
0.15.599.3.12 relocateClauses() void smtrat::VarSchedulerBase::relocateClauses (
 std::function< void(Minisat::CRef &) >) [inline], [inherited]
```

## 0.15.599.4 Field Documentation

```
0.15.599.4.1 abstractLiteral std::function<const Minisat::Lit(const FormulaT&) > smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]
```

```
0.15.599.4.2 abstractVariable std::function< Minisat::Var(const FormulaT&) > smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]
```

```
0.15.599.4.3 carlVar std::function<carl::Variable(Minisat::Var) > smtrat::VarSchedulerMcsatBase::carlVar [protected], [inherited]
```

```
0.15.599.4.4 currentlySatisfiedByBackend std::function<unsigned(const FormulaT&) > smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]
```

```
0.15.599.4.5 currentModel std::function<Model() > smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]
```

```
0.15.599.4.6 getActivity std::function<double(Minisat::Var) > smtrat::VarSchedulerBase::getActivity [protected], [inherited]
```

```
0.15.599.4.7 getBoolLitValue std::function<Minisat::lbool(Minisat::Lit) > smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]
```

```
0.15.599.4.8 getBoolVarValue std::function<Minisat::lbool(Minisat::Var) > smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]
```

```
0.15.599.4.9 getClause std::function<const Minisat::Clause&(Minisat::CRef) > smtrat::VarSchedulerBase::getClause [protected], [inherited]
```

```
0.15.599.4.10 getPolarity std::function<char(Minisat::Var) > smtrat::VarSchedulerBase::getPolarity [protected], [inherited]
```

```
0.15.599.4.11 isAbstractedFormula std::function<bool(const FormulaT&) > smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]
```

**0.15.599.4.12 isBoolValueUndef** std::function<bool ([Minisat::Var](#))> smtrat::VarSchedulerBase::::isBoolValueUndef [protected], [inherited]

**0.15.599.4.13 isDecisionVar** std::function<bool ([Minisat::Var](#))> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

**0.15.599.4.14 isTheoryAbstraction** std::function<bool ([Minisat::Var](#))> smtrat::VarSchedulerBase::::isTheoryAbstraction [protected], [inherited]

**0.15.599.4.15 isTheoryVar** std::function<bool ([Minisat::Var](#))> smtrat::VarSchedulerMcsatBase::::isTheoryVar [protected], [inherited]

**0.15.599.4.16 minisatVar** std::function<[Minisat::Var](#)([carl::Variable](#))> smtrat::VarScheduler::McsatBase::minisatVar [protected], [inherited]

**0.15.599.4.17 reabstractLiteral** std::function<const [FormulaT](#)& ([Minisat::Lit](#))> smtrat::VarScheduler::Base::reabstractLiteral [protected], [inherited]

**0.15.599.4.18 reabstractVariable** std::function<const [FormulaT](#)& ([Minisat::Var](#))> smtrat::VarScheduler::SchedulerBase::reabstractVariable [protected], [inherited]

**0.15.599.4.19 setPolarity** std::function<void ([Minisat::Var](#), bool)> smtrat::VarSchedulerBase::::setPolarity [protected], [inherited]

## 0.15.600 smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities > Class Template Reference

Decides only Constraints occurring in clauses that are univariate in the current theory variable while the theory ordering is static.

```
#include <VarSchedulerMcsat.h>
```

### Public Member Functions

- void [attachClause](#) ([Minisat::CRef](#) cl)
- void [detachClause](#) ([Minisat::CRef](#) cl)
- void [relocateClauses](#) (std::function< void([Minisat::CRef](#) &)> relocate)
- template<typename BaseModule >  
  [VarSchedulerMcsatUnivariateClausesOnly](#) (BaseModule &baseModule)
- void [rebuild](#) ()
- void [insert](#) ([Minisat::Var](#) var)
- [Minisat::Lit](#) [pop](#) ()
- bool [empty](#) ()
- void [print](#) () const
- void [increaseActivity](#) ([Minisat::Var](#) var)
- void [decreaseActivity](#) ([Minisat::Var](#) var)
- void [rebuildActivities](#) ()
- template<typename Constraints >  
  void [rebuildTheoryVars](#) (const Constraints &c)

## Protected Attributes

- std::function< bool(Minisat::Var) > isTheoryVar
- std::function< carl::Variable(Minisat::Var) > carlVar
- std::function< Minisat::Var(carl::Variable) > minisatVar
- std::function< Model() > currentModel
- std::function< double(Minisat::Var) > getActivity
- std::function< char(Minisat::Var) > getPolarity
- std::function< void(Minisat::Var, bool) > setPolarity
- std::function< bool(Minisat::Var) > isDecisionVar
- std::function< bool(Minisat::Var) > isBoolValueUndef
- std::function< bool(Minisat::Var) > isTheoryAbstraction
- std::function< const FormulaT &(Minisat::Var) > reabstractVariable
- std::function< const FormulaT &(Minisat::Lit) > reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef) > getClause
- std::function< Minisat::lbool(Minisat::Var) > getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit) > getBoolLitValue
- std::function< unsigned(const FormulaT &) > currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &) > abstractVariable
- std::function< const Minisat::Lit(const FormulaT &) > abstractLiteral
- std::function< bool(const FormulaT &) > isAbstractedFormula

### 0.15.600.1 Detailed Description

```
template<typename TheoryScheduler, bool respectActivities>
class smrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >
```

Decides only Constraints occurring in clauses that are univariate in the current theory variable while the theory ordering is static.

This corresponds to the original NLSAT strategy.

### 0.15.600.2 Constructor & Destructor Documentation

```
0.15.600.2.1 VarSchedulerMcsatUnivariateClausesOnly() template<typename TheoryScheduler , bool
respectActivities>
template<typename BaseModule >
smrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::VarSchedulerMcsatUnivar
(
 BaseModule & baseModule) [inline], [explicit]
```

### 0.15.600.3 Member Function Documentation

```
0.15.600.3.1 attachClause() template<typename TheoryScheduler , bool respectActivities>
void smrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::attachClause (
 Minisat::CRef cl) [inline]
```

```
0.15.600.3.2 decreaseActivity() template<typename TheoryScheduler , bool respectActivities>
void smrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::decreaseActivity (
 Minisat::Var var) [inline]
```

```
0.15.600.3.3 detachClause() template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::detachClause (
 Minisat::CRef cl) [inline]
```

```
0.15.600.3.4 empty() template<typename TheoryScheduler , bool respectActivities>
bool smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::empty () [inline]
```

```
0.15.600.3.5 increaseActivity() template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::increaseActivity (
 Minisat::Var var) [inline]
```

```
0.15.600.3.6 insert() template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::insert (
 Minisat::Var var) [inline]
```

```
0.15.600.3.7 pop() template<typename TheoryScheduler , bool respectActivities>
Minisat::Lit smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::pop () [inline]
```

```
0.15.600.3.8 print() template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::print () const [inline]
```

```
0.15.600.3.9 rebuild() template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::rebuild () [inline]
```

```
0.15.600.3.10 rebuildActivities() template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::rebuildActivities () [inline]
```

```
0.15.600.3.11 rebuildTheoryVars() template<typename TheoryScheduler , bool respectActivities>
template<typename Constraints >
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::rebuildTheoryVars (
 const Constraints & c) [inline]
```

```
0.15.600.3.12 relocateClauses() template<typename TheoryScheduler , bool respectActivities>
void smtrat::VarSchedulerMcsatUnivariateClausesOnly< TheoryScheduler, respectActivities >::relocateClauses (
 std::function< void(Minisat::CRef &) > relocate) [inline]
```

#### 0.15.600.4 Field Documentation

**0.15.600.4.1 abstractLiteral** std::function<const `Minisat::Lit`(const `FormulaT&`)> smtrat::Var<  
SchedulerBase::abstractLiteral [protected], [inherited]

**0.15.600.4.2 abstractVariable** std::function< `Minisat::Var`(const `FormulaT&`)> smtrat::VarScheduler<  
Base::abstractVariable [protected], [inherited]

**0.15.600.4.3 carlVar** std::function<carl::Variable(`Minisat::Var`)> smtrat::VarSchedulerMcsat<  
Base::carlVar [protected], [inherited]

**0.15.600.4.4 currentlySatisfiedByBackend** std::function<unsigned(const `FormulaT&`)> smtrat::Var<  
SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.600.4.5 currentModel** std::function<`Model`()> smtrat::VarSchedulerMcsatBase::currentModel  
[protected], [inherited]

**0.15.600.4.6 getActivity** std::function<double(`Minisat::Var`)> smtrat::VarSchedulerBase::get<  
Activity [protected], [inherited]

**0.15.600.4.7 getBoolLitValue** std::function<`Minisat::lbool`(`Minisat::Lit`)> smtrat::VarScheduler<  
Base::getBoolLitValue [protected], [inherited]

**0.15.600.4.8 getBoolVarValue** std::function<`Minisat::lbool`(`Minisat::Var`)> smtrat::VarScheduler<  
Base::getBoolVarValue [protected], [inherited]

**0.15.600.4.9 getClause** std::function<const `Minisat::Clause&`(`Minisat::CRef`)> smtrat::Var<  
SchedulerBase::getClause [protected], [inherited]

**0.15.600.4.10 getPolarity** std::function<char(`Minisat::Var`)> smtrat::VarSchedulerBase::get<  
Polarity [protected], [inherited]

**0.15.600.4.11 isAbstractedFormula** std::function<bool(const `FormulaT&`)> smtrat::VarScheduler<  
Base::isAbstractedFormula [protected], [inherited]

**0.15.600.4.12 isBoolValueUndef** std::function<bool(`Minisat::Var`)> smtrat::VarSchedulerBase<  
::isBoolValueUndef [protected], [inherited]

**0.15.600.4.13 isDecisionVar** std::function<bool(`Minisat::Var`)> smtrat::VarSchedulerBase::is<  
DecisionVar [protected], [inherited]

**0.15.600.4.14 isTheoryAbstraction** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]

**0.15.600.4.15 isTheoryVar** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]

**0.15.600.4.16 minisatVar** std::function<[Minisat::Var](#)(carl::Variable)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]

**0.15.600.4.17 reabstractLiteral** std::function<const [FormulaT](#)&([Minisat::Lit](#))> smtrat::VarScheduler::reabstractLiteral [protected], [inherited]

**0.15.600.4.18 reabstractVariable** std::function<const [FormulaT](#)&([Minisat::Var](#))> smtrat::VarScheduler::reabstractVariable [protected], [inherited]

**0.15.600.4.19 setPolarity** std::function<void([Minisat::Var](#), bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]

## 0.15.601 smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot > Class Template Reference

Decides only Constraints univariate in the current theory variable while the theory ordering is static.

```
#include <VarSchedulerMcsat.h>
```

### Public Member Functions

- template<typename BaseModule >  
  [VarSchedulerMcsatUnivariateConstraintsOnly](#) (BaseModule &baseModule)
- void [rebuild](#) ()
- void [insert](#) ([Minisat::Var](#) var)
- [Minisat::Lit](#) [pop](#) ()
- bool [empty](#) ()
- void [print](#) () const
- void [increaseActivity](#) ([Minisat::Var](#) var)
- void [decreaseActivity](#) ([Minisat::Var](#) var)
- void [rebuildActivities](#) ()
- template<typename Constraints >  
  void [rebuildTheoryVars](#) (const Constraints &c)
- void [attachClause](#) ([Minisat::CRef](#))
- void [detachClause](#) ([Minisat::CRef](#))
- void [relocateClauses](#) (std::function< void([Minisat::CRef](#) &)>)

### Protected Attributes

- std::function< bool([Minisat::Var](#))> [isTheoryVar](#)
- std::function< carl::Variable([Minisat::Var](#))> [carlVar](#)
- std::function< [Minisat::Var](#)(carl::Variable)> [minisatVar](#)
- std::function< [Model\(\)](#)> [currentModel](#)
- std::function< double([Minisat::Var](#))> [getActivity](#)
- std::function< char([Minisat::Var](#))> [getPolarity](#)
- std::function< void([Minisat::Var](#), bool)> [setPolarity](#)

- std::function< bool([Minisat::Var](#))> [isDecisionVar](#)
- std::function< bool([Minisat::Var](#))> [isBoolValueUndef](#)
- std::function< bool([Minisat::Var](#))> [isTheoryAbstraction](#)
- std::function< const [FormulaT](#) &([Minisat::Var](#))> [reabstractVariable](#)
- std::function< const [FormulaT](#) &([Minisat::Lit](#))> [reabstractLiteral](#)
- std::function< const [Minisat::Clause](#) &([Minisat::CRef](#))> [getClause](#)
- std::function< [Minisat::lbool](#)([Minisat::Var](#))> [getBoolVarValue](#)
- std::function< [Minisat::lbool](#)([Minisat::Lit](#))> [getBoolLitValue](#)
- std::function< unsigned(const [FormulaT](#) &)> [currentlySatisfiedByBackend](#)
- std::function< [Minisat::Var](#)(const [FormulaT](#) &)> [abstractVariable](#)
- std::function< const [Minisat::Lit](#)(const [FormulaT](#) &)> [abstractLiteral](#)
- std::function< bool(const [FormulaT](#) &)> [isAbstractedFormula](#)

### 0.15.601.1 Detailed Description

```
template<int lookahead, mcsat::VariableOrdering vot>
class smrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >
```

Decides only Constraints univariate in the current theory variable while the theory ordering is static.

### 0.15.601.2 Constructor & Destructor Documentation

#### 0.15.601.2.1 VarSchedulerMcsatUnivariateConstraintsOnly()

```
template<int lookahead, mcsat::VariableOrdering vot>
smrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::VarSchedulerMcsatUnivariateConstraintsOnly()
(
 BaseModule & baseModule) [inline], [explicit]
```

### 0.15.601.3 Member Function Documentation

#### 0.15.601.3.1 attachClause()

```
void smrat::VarSchedulerBase::attachClause (
 Minisat::CRef) [inline], [inherited]
```

#### 0.15.601.3.2 decreaseActivity()

```
void smrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::decreaseActivity (
 Minisat::Var var) [inline]
```

#### 0.15.601.3.3 detachClause()

```
void smrat::VarSchedulerBase::detachClause (
 Minisat::CRef) [inline], [inherited]
```

#### 0.15.601.3.4 empty()

```
bool smrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::empty () [inline]
```

#### 0.15.601.3.5 increaseActivity()

```
void smrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::increaseActivity (
 Minisat::Var var) [inline]
```

**0.15.601.3.6 `insert()`** template<int lookahead, mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::insert (  
Minisat::Var var) [inline]

**0.15.601.3.7 `pop()`** template<int lookahead, mcsat::VariableOrdering vot>  
Minisat::Lit smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::pop ()  
[inline]

**0.15.601.3.8 `print()`** template<int lookahead, mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::print () const  
[inline]

**0.15.601.3.9 `rebuild()`** template<int lookahead, mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuild () [inline]

**0.15.601.3.10 `rebuildActivities()`** template<int lookahead, mcsat::VariableOrdering vot>  
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuildActivities ()  
[inline]

**0.15.601.3.11 `rebuildTheoryVars()`** template<int lookahead, mcsat::VariableOrdering vot>  
template<typename Constraints >  
void smtrat::VarSchedulerMcsatUnivariateConstraintsOnly< lookahead, vot >::rebuildTheoryVars (  
const Constraints & c) [inline]

**0.15.601.3.12 `relocateClauses()`** void smtrat::VarSchedulerBase::relocateClauses (  
std::function< void(Minisat::CRef &) > ) [inline], [inherited]

## 0.15.601.4 Field Documentation

**0.15.601.4.1 `abstractLiteral`** std::function<const Minisat::Lit (const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]

**0.15.601.4.2 `abstractVariable`** std::function< Minisat::Var (const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]

**0.15.601.4.3 `carlVar`** std::function<carl::Variable(Minisat::Var)> smtrat::VarSchedulerMcsatBase::carlVar [protected], [inherited]

**0.15.601.4.4 `currentlySatisfiedByBackend`** std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.601.4.5 `currentModel`** std::function<Model ()> smtrat::VarSchedulerMcsatBase::currentModel [protected], [inherited]

**0.15.601.4.6 `getActivity`** std::function<double(*Minisat::Var*)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]

**0.15.601.4.7 `getBoolLitValue`** std::function<*Minisat::lbool*(*Minisat::Lit*)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

**0.15.601.4.8 `getBoolVarValue`** std::function<*Minisat::lbool*(*Minisat::Var*)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

**0.15.601.4.9 `getClause`** std::function<const *Minisat::Clause*&(*Minisat::CRef*)> smtrat::VarSchedulerBase::getClause [protected], [inherited]

**0.15.601.4.10 `getPolarity`** std::function<char(*Minisat::Var*)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

**0.15.601.4.11 `isAbstractedFormula`** std::function<bool(const *FormulaT*&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

**0.15.601.4.12 `isBoolValueUndef`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]

**0.15.601.4.13 `isDecisionVar`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

**0.15.601.4.14 `isTheoryAbstraction`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]

**0.15.601.4.15 `isTheoryVar`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerMcsatBase::isTheoryVar [protected], [inherited]

**0.15.601.4.16 `minisatVar`** std::function<*Minisat::Var*(*carl::Variable*)> smtrat::VarSchedulerMcsatBase::minisatVar [protected], [inherited]

**0.15.601.4.17 `reabstractionLiteral`** std::function<const *FormulaT*&(*Minisat::Lit*)> smtrat::VarSchedulerBase::reabstractionLiteral [protected], [inherited]

**0.15.601.4.18 `reabstractionVariable`** std::function<const *FormulaT*&(*Minisat::Var*)> smtrat::VarSchedulerBase::reabstractionVariable [protected], [inherited]

**0.15.601.4.19 `setPolarity`** std::function<void(*Minisat::Var*,*bool*)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]

## 0.15.602 smtrat::VarSchedulerMinisat Class Reference

Minisat's activity-based variable scheduling.

```
#include <VarScheduler.h>
```

### Public Member Functions

- template<typename BaseModule >  
  **VarSchedulerMinisat** (BaseModule &baseModule, std::function< bool([Minisat::Var](#), [Minisat::Var](#))> cmp)
- template<typename BaseModule >  
  **VarSchedulerMinisat** (BaseModule &baseModule)
- void **rebuild** ()
- void **insert** ([Minisat::Var](#) var)
- [Minisat::Var](#) **top** ()
- [Minisat::Lit](#) **pop** ()
- bool **empty** ()
- void **print** () const
- void **increaseActivity** ([Minisat::Var](#) var)
- void **decreaseActivity** ([Minisat::Var](#) var)
- void **rebuildActivities** ()
- template<typename Constraints >  
  void **rebuildTheoryVars** (const Constraints &)
- void **attachClause** ([Minisat::CRef](#))
- void **detachClause** ([Minisat::CRef](#))
- void **relocateClauses** (std::function< void([Minisat::CRef](#) &)>)

### Protected Member Functions

- bool **valid** ([Minisat::Var](#) var)

### Protected Attributes

- std::function< double([Minisat::Var](#))> **getActivity**
- std::function< char([Minisat::Var](#))> **getPolarity**
- std::function< void([Minisat::Var](#), bool)> **setPolarity**
- std::function< bool([Minisat::Var](#))> **isDecisionVar**
- std::function< bool([Minisat::Var](#))> **isBoolValueUndef**
- std::function< bool([Minisat::Var](#))> **isTheoryAbstraction**
- std::function< const [FormulaT](#) &([Minisat::Var](#))> **reabstractVariable**
- std::function< const [FormulaT](#) &([Minisat::Lit](#))> **reabstractLiteral**
- std::function< const [Minisat::Clause](#) &([Minisat::CRef](#))> **getClause**
- std::function< [Minisat::lbool](#)([Minisat::Var](#))> **getBoolVarValue**
- std::function< [Minisat::lbool](#)([Minisat::Lit](#))> **getBoolLitValue**
- std::function< unsigned(const [FormulaT](#) &)> **currentlySatisfiedByBackend**
- std::function< [Minisat::Var](#)(const [FormulaT](#) &)> **abstractVariable**
- std::function< const [Minisat::Lit](#)(const [FormulaT](#) &)> **abstractLiteral**
- std::function< bool(const [FormulaT](#) &)> **isAbstractedFormula**

### 0.15.602.1 Detailed Description

Minisat's activity-based variable scheduling.

### 0.15.602.2 Constructor & Destructor Documentation

**0.15.602.2.1 VarSchedulerMinisat()** [1/2] template<typename BaseModule >  
smrat::VarSchedulerMinisat::VarSchedulerMinisat (   
    BaseModule & baseModule,  
    std::function< bool([Minisat::Var](#), [Minisat::Var](#))> cmp ) [inline], [explicit]

**0.15.602.2.2 VarSchedulerMinisat()** [2/2] template<typename BaseModule >  
smrat::VarSchedulerMinisat::VarSchedulerMinisat (   
    BaseModule & baseModule ) [inline], [explicit]

### 0.15.602.3 Member Function Documentation

**0.15.602.3.1 attachClause()** void smrat::VarSchedulerBase::attachClause (   
    [Minisat::CRef](#) ) [inline], [inherited]

**0.15.602.3.2 decreaseActivity()** void smrat::VarSchedulerMinisat::decreaseActivity (   
    [Minisat::Var](#) var ) [inline]

**0.15.602.3.3 detachClause()** void smrat::VarSchedulerBase::detachClause (   
    [Minisat::CRef](#) ) [inline], [inherited]

**0.15.602.3.4 empty()** bool smrat::VarSchedulerMinisat::empty ( ) [inline]

**0.15.602.3.5 increaseActivity()** void smrat::VarSchedulerMinisat::increaseActivity (   
    [Minisat::Var](#) var ) [inline]

**0.15.602.3.6 insert()** void smrat::VarSchedulerMinisat::insert (   
    [Minisat::Var](#) var ) [inline]

**0.15.602.3.7 pop()** [Minisat::Lit](#) smrat::VarSchedulerMinisat::pop ( ) [inline]

**0.15.602.3.8 print()** void smrat::VarSchedulerMinisat::print ( ) const [inline]

**0.15.602.3.9 rebuild()** void smrat::VarSchedulerMinisat::rebuild ( ) [inline]

**0.15.602.3.10 rebuildActivities()** void smrat::VarSchedulerMinisat::rebuildActivities ( ) [inline]

**0.15.602.3.11 rebuildTheoryVars()** template<typename Constraints >  
void smrat::VarSchedulerBase::rebuildTheoryVars (   
    const Constraints & ) [inline], [inherited]

**0.15.602.3.12 relocateClauses()** void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &) > ) [inline], [inherited]

**0.15.602.3.13 top()** Minisat::Var smtrat::VarSchedulerMinisat::top () [inline]

**0.15.602.3.14 valid()** bool smtrat::VarSchedulerMinisat::valid ( Minisat::Var var ) [inline], [protected]

#### 0.15.602.4 Field Documentation

**0.15.602.4.1 abstractLiteral** std::function<const Minisat::Lit (const FormulaT&) > smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]

**0.15.602.4.2 abstractVariable** std::function< Minisat::Var (const FormulaT&) > smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]

**0.15.602.4.3 currentlySatisfiedByBackend** std::function<unsigned (const FormulaT&) > smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.602.4.4 getActivity** std::function<double (Minisat::Var) > smtrat::VarSchedulerBase::getActivity [protected], [inherited]

**0.15.602.4.5 getBoolLitValue** std::function<Minisat::lbool (Minisat::Lit) > smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

**0.15.602.4.6 getBoolVarValue** std::function<Minisat::lbool (Minisat::Var) > smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

**0.15.602.4.7 getClause** std::function<const Minisat::Clause& (Minisat::CRef) > smtrat::VarSchedulerBase::getClause [protected], [inherited]

**0.15.602.4.8 getPolarity** std::function<char (Minisat::Var) > smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

**0.15.602.4.9 isAbstractedFormula** std::function<bool (const FormulaT&) > smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

**0.15.602.4.10 isBoolValueUndef** std::function<bool (Minisat::Var) > smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]

**0.15.602.4.11 `isDecisionVar`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerBase::is←  
DecisionVar [protected], [inherited]

**0.15.602.4.12 `isTheoryAbstraction`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerBase←  
::isTheoryAbstraction [protected], [inherited]

**0.15.602.4.13 `reabstractLiteral`** std::function<const *FormulaT*&(*Minisat::Lit*)> smtrat::VarScheduler←  
Base::reabstractLiteral [protected], [inherited]

**0.15.602.4.14 `reabstractVariable`** std::function<const *FormulaT*&(*Minisat::Var*)> smtrat::Var←  
SchedulerBase::reabstractVariable [protected], [inherited]

**0.15.602.4.15 `setPolarity`** std::function<void(*Minisat::Var*, bool)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]

## 0.15.603 smtrat::VarSchedulerRandom Class Reference

Random scheduler.

```
#include <VarScheduler.h>
```

### Public Member Functions

- template<typename BaseModule>  
`VarSchedulerRandom` (BaseModule &baseModule)
- void `rebuild` ()
- void `insert` (*Minisat::Var* var)
- *Minisat::Lit* `pop` ()
- bool `empty` ()
- void `print` () const
- void `increaseActivity` (*Minisat::Var*)
- void `decreaseActivity` (*Minisat::Var*)
- void `rebuildActivities` ()
- template<typename Constraints>  
void `rebuildTheoryVars` (const Constraints &)
- void `attachClause` (*Minisat::CRef*)
- void `detachClause` (*Minisat::CRef*)
- void `relocateClauses` (std::function< void(*Minisat::CRef* &) >)

### Protected Member Functions

- bool `valid` (*Minisat::Var* var)

### Protected Attributes

- std::function< double(*Minisat::Var*)> `getActivity`
- std::function< char(*Minisat::Var*)> `getPolarity`
- std::function< void(*Minisat::Var*, bool)> `setPolarity`
- std::function< bool(*Minisat::Var*)> `isDecisionVar`
- std::function< bool(*Minisat::Var*)> `isBoolValueUndef`
- std::function< bool(*Minisat::Var*)> `isTheoryAbstraction`
- std::function< const *FormulaT* &(*Minisat::Var*)> `reabstractVariable`
- std::function< const *FormulaT* &(*Minisat::Lit*)> `reabstractLiteral`

- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const FormulaT &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const FormulaT &)> abstractVariable
- std::function< const Minisat::Lit(const FormulaT &)> abstractLiteral
- std::function< bool(const FormulaT &)> isAbstractedFormula

### 0.15.603.1 Detailed Description

Random scheduler.

### 0.15.603.2 Constructor & Destructor Documentation

**0.15.603.2.1 VarSchedulerRandom()** template<typename BaseModule >  
smtrat::VarSchedulerRandom::VarSchedulerRandom (

BaseModule & *baseModule* ) [inline], [explicit]

### 0.15.603.3 Member Function Documentation

**0.15.603.3.1 attachClause()** void smtrat::VarSchedulerBase::attachClause (

Minisat::CRef ) [inline], [inherited]

**0.15.603.3.2 decreaseActivity()** void smtrat::VarSchedulerBase::decreaseActivity (

Minisat::Var ) [inline], [inherited]

**0.15.603.3.3 detachClause()** void smtrat::VarSchedulerBase::detachClause (

Minisat::CRef ) [inline], [inherited]

**0.15.603.3.4 empty()** bool smtrat::VarSchedulerRandom::empty ( ) [inline]

**0.15.603.3.5 increaseActivity()** void smtrat::VarSchedulerBase::increaseActivity (

Minisat::Var ) [inline], [inherited]

**0.15.603.3.6 insert()** void smtrat::VarSchedulerRandom::insert (

Minisat::Var *var* ) [inline]

**0.15.603.3.7 pop()** Minisat::Lit smtrat::VarSchedulerRandom::pop ( ) [inline]

**0.15.603.3.8 print()** void smtrat::VarSchedulerRandom::print ( ) const [inline]

**0.15.603.3.9 rebuild()** void smtrat::VarSchedulerRandom::rebuild ( ) [inline]

**0.15.603.3.10 rebuildActivities()** void smtrat::VarSchedulerBase::rebuildActivities () [inline], [inherited]

**0.15.603.3.11 rebuildTheoryVars()** template<typename Constraints >  
void smtrat::VarSchedulerBase::rebuildTheoryVars (  
 const Constraints & ) [inline], [inherited]

**0.15.603.3.12 relocateClauses()** void smtrat::VarSchedulerBase::relocateClauses (  
 std::function< void(Minisat::CRef &) > ) [inline], [inherited]

**0.15.603.3.13 valid()** bool smtrat::VarSchedulerRandom::valid (  
 Minisat::Var var ) [inline], [protected]

## 0.15.603.4 Field Documentation

**0.15.603.4.1 abstractLiteral** std::function<const Minisat::Lit (const FormulaT&)> smtrat::VarSchedulerBase::abstractLiteral [protected], [inherited]

**0.15.603.4.2 abstractVariable** std::function< Minisat::Var(const FormulaT&)> smtrat::VarSchedulerBase::abstractVariable [protected], [inherited]

**0.15.603.4.3 currentlySatisfiedByBackend** std::function<unsigned(const FormulaT&)> smtrat::VarSchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.603.4.4 getActivity** std::function<double(Minisat::Var)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]

**0.15.603.4.5 getBoolLitValue** std::function<Minisat::lbool(Minisat::Lit)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

**0.15.603.4.6 getBoolVarValue** std::function<Minisat::lbool(Minisat::Var)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

**0.15.603.4.7 getClause** std::function<const Minisat::Clause&(Minisat::CRef)> smtrat::VarSchedulerBase::getClause [protected], [inherited]

**0.15.603.4.8 getPolarity** std::function<char(Minisat::Var)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

**0.15.603.4.9 isAbstractedFormula** std::function<bool(const FormulaT&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

**0.15.603.4.10 `isBoolValueUndef`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::::isBoolValueUdef [protected], [inherited]

**0.15.603.4.11 `isDecisionVar`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

**0.15.603.4.12 `isTheoryAbstraction`** std::function<bool([Minisat::Var](#))> smtrat::VarSchedulerBase::::isTheoryAbstraction [protected], [inherited]

**0.15.603.4.13 `reabstractLiteral`** std::function<const [FormulaT](#)&([Minisat::Lit](#))> smtrat::VarScheduler::Base::reabstractLiteral [protected], [inherited]

**0.15.603.4.14 `reabstractVariable`** std::function<const [FormulaT](#)&([Minisat::Var](#))> smtrat::VarSchedulerBase::reabstractVariable [protected], [inherited]

**0.15.603.4.15 `setPolarity`** std::function<void([Minisat::Var](#), bool)> smtrat::VarSchedulerBase::::setPolarity [protected], [inherited]

## 0.15.604 smtrat::VarSchedulerSMTTheoryGuided< theory\_conflict\_guided\_decision\_heuristic > Class Template Reference

Scheduler for SMT, implementing theory guided heuristics.

```
#include <VarScheduler.h>
```

### Public Member Functions

- template<typename BaseModule >  
[VarSchedulerSMTTheoryGuided](#) (BaseModule &baseModule)
- void [rebuild](#) ()
- void [insert](#) ([Minisat::Var](#) var)
- [Minisat::Lit](#) [pop](#) ()
- bool [empty](#) ()
- void [print](#) () const
- void [increaseActivity](#) ([Minisat::Var](#) var)
- void [decreaseActivity](#) ([Minisat::Var](#) var)
- void [rebuildActivities](#) ()
- template<typename Constraints >  
void [rebuildTheoryVars](#) (const Constraints &)
- void [attachClause](#) ([Minisat::CRef](#))
- void [detachClause](#) ([Minisat::CRef](#))
- void [relocateClauses](#) (std::function< void([Minisat::CRef](#) &)>)

### Protected Attributes

- std::function< double([Minisat::Var](#))> [getActivity](#)
- std::function< char([Minisat::Var](#))> [getPolarity](#)
- std::function< void([Minisat::Var](#), bool)> [setPolarity](#)
- std::function< bool([Minisat::Var](#))> [isDecisionVar](#)
- std::function< bool([Minisat::Var](#))> [isBoolValueUdef](#)
- std::function< bool([Minisat::Var](#))> [isTheoryAbstraction](#)
- std::function< const [FormulaT](#) &([Minisat::Var](#))> [reabstractVariable](#)

- std::function< const [FormulaT](#) &(Minisat::Lit)> reabstractLiteral
- std::function< const Minisat::Clause &(Minisat::CRef)> getClause
- std::function< Minisat::lbool(Minisat::Var)> getBoolVarValue
- std::function< Minisat::lbool(Minisat::Lit)> getBoolLitValue
- std::function< unsigned(const [FormulaT](#) &)> currentlySatisfiedByBackend
- std::function< Minisat::Var(const [FormulaT](#) &)> abstractVariable
- std::function< const Minisat::Lit(const [FormulaT](#) &)> abstractLiteral
- std::function< bool(const [FormulaT](#) &)> isAbstractedFormula

#### 0.15.604.1 Detailed Description

```
template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_decision_heuristic>
class smrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >
```

Scheduler for SMT, implementing theory guided heuristics.

#### 0.15.604.2 Constructor & Destructor Documentation

```
0.15.604.2.1 VarSchedulerSMTTheoryGuided() template<TheoryGuidedDecisionHeuristicLevel theory←
_ conflict_guided_decision_heuristic>
template<typename BaseModule >
smrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::VarSchedulerSMTTheoryGuided
(
 BaseModule & baseModule) [inline], [explicit]
```

#### 0.15.604.3 Member Function Documentation

```
0.15.604.3.1 attachClause() void smrat::VarSchedulerBase::attachClause (
 Minisat::CRef) [inline], [inherited]
```

```
0.15.604.3.2 decreaseActivity() template<TheoryGuidedDecisionHeuristicLevel theory_conflict_←
guided_decision_heuristic>
void smrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::decrease←
Activity (
 Minisat::Var var) [inline]
```

```
0.15.604.3.3 detachClause() void smrat::VarSchedulerBase::detachClause (
 Minisat::CRef) [inline], [inherited]
```

```
0.15.604.3.4 empty() template<TheoryGuidedDecisionHeuristicLevel theory_conflict_guided_←
decision_heuristic>
bool smrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::empty (
) [inline]
```

```
0.15.604.3.5 increaseActivity() template<TheoryGuidedDecisionHeuristicLevel theory_conflict_←
guided_decision_heuristic>
void smrat::VarSchedulerSMTTheoryGuided< theory_conflict_guided_decision_heuristic >::increase←
Activity (
 Minisat::Var var) [inline]
```

**0.15.604.3.6 `insert()`** template<TheoryGuidedDecisionHeuristicLevel theory\_conflict\_guided\_<  
decision\_heuristic>  
void smtrat::VarSchedulerSMTTheoryGuided< theory\_conflict\_guided\_decision\_heuristic >::insert  
(  
    Minisat::Var var ) [inline]

**0.15.604.3.7 `pop()`** template<TheoryGuidedDecisionHeuristicLevel theory\_conflict\_guided\_decision\_<  
\_heuristic>  
Minisat::Lit smtrat::VarSchedulerSMTTheoryGuided< theory\_conflict\_guided\_decision\_heuristic  
>::pop ( ) [inline]

**0.15.604.3.8 `print()`** template<TheoryGuidedDecisionHeuristicLevel theory\_conflict\_guided\_decision\_<  
\_heuristic>  
void smtrat::VarSchedulerSMTTheoryGuided< theory\_conflict\_guided\_decision\_heuristic >::print ( ) const [inline]

**0.15.604.3.9 `rebuild()`** template<TheoryGuidedDecisionHeuristicLevel theory\_conflict\_guided\_<  
decision\_heuristic>  
void smtrat::VarSchedulerSMTTheoryGuided< theory\_conflict\_guided\_decision\_heuristic >::rebuild  
( ) [inline]

**0.15.604.3.10 `rebuildActivities()`** template<TheoryGuidedDecisionHeuristicLevel theory\_conflict\_<  
guided\_decision\_heuristic>  
void smtrat::VarSchedulerSMTTheoryGuided< theory\_conflict\_guided\_decision\_heuristic >::rebuild\_<  
Activities ( ) [inline]

**0.15.604.3.11 `rebuildTheoryVars()`** template<typename Constraints >  
void smtrat::VarSchedulerBase::rebuildTheoryVars ( const Constraints & ) [inline], [inherited]

**0.15.604.3.12 `relocateClauses()`** void smtrat::VarSchedulerBase::relocateClauses ( std::function< void(Minisat::CRef &) > ) [inline], [inherited]

## 0.15.604.4 Field Documentation

**0.15.604.4.1 `abstractLiteral`** std::function<const Minisat::Lit (const FormulaT&)> smtrat::Var\_<  
SchedulerBase::abstractLiteral [protected], [inherited]

**0.15.604.4.2 `abstractVariable`** std::function< Minisat::Var (const FormulaT&)> smtrat::VarScheduler\_<  
Base::abstractVariable [protected], [inherited]

**0.15.604.4.3 `currentlySatisfiedByBackend`** std::function<unsigned(const FormulaT&)> smtrat::Var\_<  
SchedulerBase::currentlySatisfiedByBackend [protected], [inherited]

**0.15.604.4.4 `getActivity`** std::function<double(*Minisat::Var*)> smtrat::VarSchedulerBase::getActivity [protected], [inherited]

**0.15.604.4.5 `getBoolLitValue`** std::function<*Minisat::lbool*(*Minisat::Lit*)> smtrat::VarSchedulerBase::getBoolLitValue [protected], [inherited]

**0.15.604.4.6 `getBoolVarValue`** std::function<*Minisat::lbool*(*Minisat::Var*)> smtrat::VarSchedulerBase::getBoolVarValue [protected], [inherited]

**0.15.604.4.7 `getClause`** std::function<const *Minisat::Clause*&(*Minisat::CRef*)> smtrat::VarSchedulerBase::getClause [protected], [inherited]

**0.15.604.4.8 `getPolarity`** std::function<char(*Minisat::Var*)> smtrat::VarSchedulerBase::getPolarity [protected], [inherited]

**0.15.604.4.9 `isAbstractedFormula`** std::function<bool(const *FormulaT*&)> smtrat::VarSchedulerBase::isAbstractedFormula [protected], [inherited]

**0.15.604.4.10 `isBoolValueUndef`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerBase::isBoolValueUndef [protected], [inherited]

**0.15.604.4.11 `isDecisionVar`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerBase::isDecisionVar [protected], [inherited]

**0.15.604.4.12 `isTheoryAbstraction`** std::function<bool(*Minisat::Var*)> smtrat::VarSchedulerBase::isTheoryAbstraction [protected], [inherited]

**0.15.604.4.13 `reabstractionLiteral`** std::function<const *FormulaT*&(*Minisat::Lit*)> smtrat::VarSchedulerBase::reabstractionLiteral [protected], [inherited]

**0.15.604.4.14 `reabstractionVariable`** std::function<const *FormulaT*&(*Minisat::Var*)> smtrat::VarSchedulerBase::reabstractionVariable [protected], [inherited]

**0.15.604.4.15 `setPolarity`** std::function<void(*Minisat::Var*,*bool*)> smtrat::VarSchedulerBase::setPolarity [protected], [inherited]

## 0.15.605 *Minisat::vec< T >* Class Template Reference

#include <Vec.h>

## Public Member Functions

- `vec ()`
- `vec (int size)`
- `vec (int size, const T &pad)`
- `vec (vec< T > &&_toMove)`
- `~vec ()`
- `vec< T > & operator= (vec< T > &&_toMove)`
- `operator T* (void)`
- `int size (void) const`
- `void shrink (int nelems)`
- `void shrink_ (int nelems)`
- `int capacity (void) const`
- `void capacity (int min_cap)`
- `void growTo (int size)`
- `void growTo (int size, const T &pad)`
- `void clear (bool dealloc=false)`
- `void push (void)`
- `void push (const T &elem)`
- `void push_ (const T &elem)`
- `void pop (void)`
- `const T & last (void) const`
- `T & last (void)`
- `const T & operator[] (int index) const`
- `T & operator[] (int index)`
- `void copyTo (vec< T > &copy) const`
- `void moveTo (vec< T > &dest)`

### 0.15.605.1 Constructor & Destructor Documentation

**0.15.605.1.1 `vec()` [1/4]** template<class T >  
`Minisat::vec< T >::vec ( )` [inline]

**0.15.605.1.2 `vec()` [2/4]** template<class T >  
`Minisat::vec< T >::vec (`  
    `int size )` [inline], [explicit]

**0.15.605.1.3 `vec()` [3/4]** template<class T >  
`Minisat::vec< T >::vec (`  
    `int size,`  
    `const T & pad )` [inline]

**0.15.605.1.4 `vec()` [4/4]** template<class T >  
`Minisat::vec< T >::vec (`  
    `vec< T > && _toMove )` [inline]

**0.15.605.1.5 `~vec()`** template<class T >  
`Minisat::vec< T >::~vec ( )` [inline]

## 0.15.605.2 Member Function Documentation

**0.15.605.2.1 capacity() [1/2]** template<class T >  
void Minisat::vec< T >::capacity ( int min\_cap )

**0.15.605.2.2 capacity() [2/2]** template<class T >  
int Minisat::vec< T >::capacity ( void ) const [inline]

**0.15.605.2.3 clear()** template<class T >  
void Minisat::vec< T >::clear ( bool dealloc = false )

**0.15.605.2.4 copyTo()** template<class T >  
void Minisat::vec< T >::copyTo ( vec< T > & copy ) const [inline]

**0.15.605.2.5 growTo() [1/2]** template<class T >  
void Minisat::vec< T >::growTo ( int size )

**0.15.605.2.6 growTo() [2/2]** template<class T >  
void Minisat::vec< T >::growTo ( int size, const T & pad )

**0.15.605.2.7 last() [1/2]** template<class T >  
T& Minisat::vec< T >::last ( void ) [inline]

**0.15.605.2.8 last() [2/2]** template<class T >  
const T& Minisat::vec< T >::last ( void ) const [inline]

**0.15.605.2.9 moveTo()** template<class T >  
void Minisat::vec< T >::moveTo ( vec< T > & dest ) [inline]

**0.15.605.2.10 operator T\*()** template<class T >  
Minisat::vec< T >::operator T\* ( void ) [inline]

```
0.15.605.2.11 operator=() template<class T >
vec<T>& Minisat::vec< T >::operator= (
 vec< T > && _toMove) [inline]
```

```
0.15.605.2.12 operator[](1/2) template<class T >
T& Minisat::vec< T >::operator[] (
 int index) [inline]
```

```
0.15.605.2.13 operator[](2/2) template<class T >
const T& Minisat::vec< T >::operator[] (
 int index) const [inline]
```

```
0.15.605.2.14 pop() template<class T >
void Minisat::vec< T >::pop (
 void) [inline]
```

```
0.15.605.2.15 push() [1/2] template<class T >
void Minisat::vec< T >::push (
 const T & elem) [inline]
```

```
0.15.605.2.16 push() [2/2] template<class T >
void Minisat::vec< T >::push (
 void) [inline]
```

```
0.15.605.2.17 push_() template<class T >
void Minisat::vec< T >::push_ (
 const T & elem) [inline]
```

```
0.15.605.2.18 shrink() template<class T >
void Minisat::vec< T >::shrink (
 int nelems) [inline]
```

```
0.15.605.2.19 shrink_() template<class T >
void Minisat::vec< T >::shrink_ (
 int nelems) [inline]
```

```
0.15.605.2.20 size() template<class T >
int Minisat::vec< T >::size (
 void) const [inline]
```

## 0.15.606 smrat::parser::conversion::VectorVariantConverter< Res > Struct Template Reference

Converts a vector of variants to a vector of some type using the [Converter](#) class.  
#include <Conversions.h>

## Public Types

- `typedef Res result_type`

## Public Member Functions

- `template<BOOST_VARIANT_ENUM_PARAMS(typename T)>`  
`bool operator() (const std::vector< boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)>> &v, std::vector< Res > &result) const`
- `template<BOOST_VARIANT_ENUM_PARAMS(typename T)>`  
`bool operator() (const std::vector< boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)>> &v, std::vector< Res > &result, TheoryError &errors) const`

### 0.15.606.1 Detailed Description

```
template<typename Res>
struct smtrat::parser::conversion::VectorVariantConverter< Res >
```

Converts a vector of variants to a vector of some type using the [Converter](#) class.

### 0.15.606.2 Member Typedef Documentation

```
0.15.606.2.1 result_type template<typename Res >
typedef Res smtrat::parser::conversion::VectorVariantConverter< Res >::result_type
```

### 0.15.606.3 Member Function Documentation

```
0.15.606.3.1 operator() [1/2] template<typename Res >
template<BOOST_VARIANT_ENUM_PARAMS(typename T) >
bool smtrat::parser::conversion::VectorVariantConverter< Res >::operator() (
 const std::vector< boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)>> & v,
 std::vector< Res > & result) const [inline]
```

```
0.15.606.3.2 operator() [2/2] template<typename Res >
template<BOOST_VARIANT_ENUM_PARAMS (typename T) >
bool smtrat::parser::conversion::VectorVariantConverter< Res >::operator() (
 const std::vector< boost::variant< BOOST_VARIANT_ENUM_PARAMS(T)>> & v,
 std::vector< Res > & result,
 TheoryError & errors) const [inline]
```

## 0.15.607 smtrat::subtropical::Vertex Struct Reference

Represents a term of an original constraint and assigns him a rating variable if a weak separator is searched.  
`#include <Subtropical.h>`

## Public Member Functions

- `Vertex (const TermT &term)`
- `carl::Variable rating () const`

## Data Fields

- const [Rational coefficient](#)  
*Coefficient of the assigned term.*
- const [carl::Monomial::Arg monomial](#)  
*Monomial of the assigned term.*
- std::optional< [carl::Variable](#) > [m\\_rating](#)  
*Rating variable of the term for a weak separator.*

### 0.15.607.1 Detailed Description

Represents a term of an original constraint and assigns him a rating variable if a weak separator is searched.

### 0.15.607.2 Constructor & Destructor Documentation

**0.15.607.2.1 Vertex()** `smtrat::subtropical::Vertex::Vertex (`  
  `const TermT & term ) [inline]`

### 0.15.607.3 Member Function Documentation

**0.15.607.3.1 rating()** `carl::Variable smtrat::subtropical::Vertex::rating ( ) const [inline]`

### 0.15.607.4 Field Documentation

**0.15.607.4.1 coefficient** `const Rational smtrat::subtropical::Vertex::coefficient`  
Coefficient of the assigned term.

**0.15.607.4.2 m\_rating** `std::optional<carl::Variable> smtrat::subtropical::Vertex::m_rating [mutable]`  
Rating variable of the term for a weak separator.

**0.15.607.4.3 monomial** `const carl::Monomial::Arg smtrat::subtropical::Vertex::monomial`  
Monomial of the assigned term.

## 0.15.608 smtrat::VSModule< Settings > Class Template Reference

```
#include <VSModule.h>
```

### Public Types

- [typedef Settings SettingsType](#)
- [enum class LemmaType : unsigned { NORMAL = 0 , PERMANENT = 1 }](#)

## Public Member Functions

- std::string **moduleName** () const
- **VModule** (const ModuleInput \*, Conditionals &, Manager \*const =NULL)
- **~VModule** ()
- bool **addCore** (ModuleInput::const\_iterator)

*The module has to take the given sub-formula of the received formula into account.*

- **Answer checkCore** ()

*Checks the received formula for consistency.*

- void **removeCore** (ModuleInput::const\_iterator)

*Removes everything related to the given sub-formula of the received formula.*

- void **updateModel** () const

*Updates the model, if the solver has detected the consistency of the received formula, beforehand.*

- void **printAll** (const std::string &\_init="", std::ostream &\_out=std::cout) const

*Prints the history to the output stream.*

- void **printFormulaConditionMap** (const std::string &\_init="", std::ostream &\_out=std::cout) const

*Prints the history to the output stream.*

- void **printRanking** (const std::string &\_init="", std::ostream &\_out=std::cout) const

*Prints the history to the output stream.*

- void **printAnswer** (const std::string &\_init="", std::ostream &\_out=std::cout) const

*Prints the answer if existent.*

- bool **inform** (const FormulaT &\_constraint)

*Informs the module about the given constraint.*

- void **deinform** (const FormulaT &\_constraint)

*The inverse of informing about a constraint.*

- virtual void **init** ()

*Informs all backends about the so far encountered constraints, which have not yet been communicated.*

- bool **add** (ModuleInput::const\_iterator \_subformula)

*The module has to take the given sub-formula of the received formula into account.*

- virtual **Answer check** (bool \_final=false, bool \_full=true, carl::Variable \_objective=carl::Variable::NO\_← VARIABLE)

*Checks the received formula for consistency.*

- virtual void **remove** (ModuleInput::const\_iterator \_subformula)

*Removes everything related to the given sub-formula of the received formula.*

- virtual void **updateAllModels** ()

*Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.*

- virtual std::list< std::vector< carl::Variable > > **getModelEqualities** () const

*Partition the variables from the current model into equivalence classes according to their assigned value.*

- unsigned **currentlySatisfiedByBackend** (const FormulaT &\_formula) const
- virtual unsigned **currentlySatisfied** (const FormulaT &) const
- bool **receivedVariable** (carl::Variable::Arg \_var) const
- **Answer solverState** () const
- std::size\_t **id** () const
- void **setId** (std::size\_t \_id)

*Sets this modules unique ID to identify itself.*

- **thread\_priority threadPriority** () const
- void **setThreadPriority** (**thread\_priority** \_threadPriority)

*Sets the priority of this module to get a thread for running its check procedure.*

- const ModuleInput \* **pReceivedFormula** () const
- const ModuleInput & **rReceivedFormula** () const
- const ModuleInput \* **pPassedFormula** () const
- const ModuleInput & **rPassedFormula** () const

- const `Model & model () const`
- const `std::vector< Model > & allModels () const`
- const `std::vector< FormulaSetT > & infeasibleSubsets () const`
- const `std::vector< Module * > & usedBackends () const`
- const `carl::FastSet< FormulaT > & constraintsToInform () const`
- const `carl::FastSet< FormulaT > & informedConstraints () const`
- void `addLemma (const FormulaT & _lemma, const LemmaType & _lt=LemmaType::NORMAL, const FormulaT & _preferredFormula=FormulaT(carl::FormulaType::TRUE))`

*Stores a lemma being a valid formula.*
- bool `hasLemmas ()`

*Checks whether this module or any of its backends provides any lemmas.*
- void `clearLemmas ()`

*Deletes all yet found lemmas.*
- const `std::vector< Lemma > & lemmas () const`
- `ModuleInput::const_iterator firstUncheckedReceivedSubformula () const`
- `ModuleInput::const_iterator firstSubformulaToPass () const`
- void `receivedFormulaChecked ()`

*Notifies that the received formulas has been checked.*
- const `smrat::Conditionals & answerFound () const`
- bool `isPreprocessor () const`
- `carl::Variable objective () const`
- bool `is_minimizing () const`
- void `excludeNotReceivedVariablesFromModel () const`

*Excludes all variables from the current model, which do not occur in the received formula.*
- void `updateLemmas ()`

*Stores all lemmas of any backend of this module in its own lemma vector.*
- void `collectTheoryPropagations ()`
- void `collectOrigins (const FormulaT & _formula, FormulasT & _origins) const`

*Collects the formulas in the given formula, which are part of the received formula.*
- void `collectOrigins (const FormulaT & _formula, FormulaSetT & _origins) const`
- bool `hasValidInfeasibleSubset () const`
- void `checkInfSubsetForMinimality (std::vector< FormulaSetT >::const_iterator _infsSubset, const std::string & _filename="smaller_muses", unsigned _maxSizeDifference=1) const`

*Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.*
- virtual `std::pair< bool, FormulaT > getReceivedFormulaSimplified ()`
- void `print (const std::string & _initiation="***") const`

*Prints everything relevant of the solver.*
- void `printReceivedFormula (const std::string & _initiation="***") const`

*Prints the vector of the received formula.*
- void `printPassedFormula (const std::string & _initiation="***") const`

*Prints the vector of passed formula.*
- void `printInfeasibleSubsets (const std::string & _initiation="***") const`

*Prints the infeasible subsets.*
- void `printModel (std::ostream & _out=std::cout) const`

*Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.*
- void `printAllModels (std::ostream & _out=std::cout)`

*Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.*

## Static Public Member Functions

- static void `freeSplittingVariable (const FormulaT & _splittingVariable)`

## Static Public Attributes

- static size\_t `mNumOfBranchVarsToStore` = 5  
*The number of different variables to consider for a probable infinite loop of branchings.*
- static std::vector< Branching > `mLastBranches` = std::vector<Branching>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0))  
*Stores the last branches in a cycle buffer.*
- static size\_t `mFirstPosInLastBranches` = 0  
*The beginning of the cyclic buffer storing the last branches.*
- static std::vector< FormulaT > `mOldSplittingVariables`  
*Reusable splitting variables.*

## Protected Member Functions

- virtual bool `informCore` (const FormulaT &\_constraint)  
*Informs the module about the given constraint.*
- virtual void `deinformCore` (const FormulaT &\_constraint)  
*The inverse of informing about a constraint.*
- bool `anAnswerFound` () const  
*Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.*
- void `clearModel` () const  
*Clears the assignment, if any was found.*
- void `clearModels` () const  
*Clears all assignments, if any was found.*
- void `cleanModel` () const  
*Substitutes variable occurrences by its model value in the model values of other variables.*
- `ModuleInput::iterator passedFormulaBegin` ()
- `ModuleInput::iterator passedFormulaEnd` ()
- void `addOrigin` (`ModuleInput::iterator` \_formula, const FormulaT &\_origin)  
*Adds the given set of formulas in the received formula to the origins of the given passed formula.*
- const FormulaT & `getOrigins` (`ModuleInput::const_iterator` \_formula) const  
*Gets the origins of the passed formula at the given position.*
- void `getOrigins` (const FormulaT &\_formula, FormulasT &\_origins) const
- void `getOrigins` (const FormulaT &\_formula, FormulaSetT &\_origins) const
- std::pair< `ModuleInput::iterator`, bool > `removeOrigin` (`ModuleInput::iterator` \_formula, const FormulaT &\_origin)  
*ptr< std::vector< FormulaT >> &\_origins)*
- std::pair< `ModuleInput::iterator`, bool > `removeOrigins` (`ModuleInput::iterator` \_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)
- void `informBackends` (const FormulaT &\_constraint)  
*Informs all backends of this module about the given constraint.*
- virtual void `addConstraintToInform` (const FormulaT &\_constraint)  
*Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.*
- std::pair< `ModuleInput::iterator`, bool > `addReceivedSubformulaToPassedFormula` (`ModuleInput::const_iterator` \_subformula)  
*Copies the given sub-formula of the received formula to the passed formula.*
- bool `originInReceivedFormula` (const FormulaT &\_origin) const
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula)  
*Adds the given formula to the passed formula with no origin.*
- std::pair< `ModuleInput::iterator`, bool > `addSubformulaToPassedFormula` (const FormulaT &\_formula, const std::shared\_ptr< std::vector< FormulaT >> &\_origins)  
*Adds the given formula to the passed formula.*

- std::pair< [ModuleInput::iterator](#), bool > [addSubformulaToPassedFormula](#) (const [FormulaT](#) &\_formula, const [FormulaT](#) &\_origin)
 

*Adds the given formula to the passed formula.*
- void [generateTrivialInfeasibleSubset](#) ()
 

*Stores the trivial infeasible subset being the set of received formulas.*
- void [receivedFormulasAsInfeasibleSubset](#) ([ModuleInput::const\\_iterator](#) \_subformula)
 

*Stores an infeasible subset consisting only of the given received formula.*
- std::vector< [FormulaT](#) >::const\_iterator [findBestOrigin](#) (const std::vector< [FormulaT](#) > &\_origins) const
 

*Copies the infeasible subsets of the passed formula.*
- std::vector< [FormulaSetT](#) > [getInfeasibleSubsets](#) (const [Module](#) &\_backend) const
 

*Get the infeasible subsets the given backend provides.*
- const [Model](#) & [backendsModel](#) () const
 

*Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.*
- void [getBackendsModel](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
- void [getBackendsAllModels](#) () const
 

*Stores all models of a backend in the list of all models of this module.*
- virtual [Answer](#) [runBackends](#) (bool \_final, bool \_full, carl::Variable \_objective)
 

*Runs the backend solvers on the passed formula.*
- virtual [Answer](#) [runBackends](#) ()
 

*Removes everything related to the sub-formula to remove from the passed formula in the backends of this module.*
- void [clearPassedFormula](#) ()
 

*Merges the two vectors of sets into the first one.*
- std::vector< [FormulaT](#) > [merge](#) (const std::vector< [FormulaT](#) > &, const std::vector< [FormulaT](#) > &) const
 

*Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (const [Poly](#) &\_polynomial, bool \_integral, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &←\_premise=std::vector< [FormulaT](#) >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, std::vector< [FormulaT](#) > &&\_premise, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false)
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- bool [branchAt](#) (carl::Variable::Arg \_var, const [Rational](#) &\_value, bool \_leftCaseWeak=true, bool \_preferLeftCase=true, bool \_useReceivedFormulaAsPremise=false, const std::vector< [FormulaT](#) > &\_premise=std::vector< [FormulaT](#) >())
 

*Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding SATModule.*
- void [splitUnequalConstraint](#) (const [FormulaT](#) &)
 

*Adds the following lemmas for the given constraint p!=0.*
- unsigned [checkModel](#) () const
 

*Gets all InformationRelevantFormulas.*
- void [addInformationRelevantFormula](#) (const [FormulaT](#) &formula)
 

*Adds a formula to the InformationRelevantFormula.*
- const std::set< [FormulaT](#) > & [getInformationRelevantFormulas](#) ()
 

*Gets all InformationRelevantFormulas.*
- bool [isLemmaLevel](#) ([LemmaLevel](#) level)
 

*Checks if current lemma level is greater or equal to given level.*

## Static Protected Member Functions

- static bool `modelsDisjoint` (const `Model` &\_modelA, const `Model` &\_modelB)

*Checks whether there is no variable assigned by both models.*

## Protected Attributes

- `std::vector<FormulaSetT> mInfeasibleSubsets`

*Stores the infeasible subsets.*

- `Manager *const mpManager`

*A reference to the manager.*

- `Model mModel`

*Stores the assignment of the current satisfiable result, if existent.*

- `std::vector<Model> mAllModels`

*Stores all satisfying assignments.*

- `bool mModelComputed`

*True, if the model has already been computed.*

- `bool mFinalCheck`

*true, if the check procedure should perform a final check which especially means not to postpone splitting decisions*

- `bool mFullCheck`

*false, if this module should avoid too expensive procedures and rather return unknown instead.*

- `carl::Variable mObjectiveVariable`

*Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.*

- `std::vector<TheoryPropagation> mTheoryPropagations`

- `std::atomic<Answer> mSolverState`

*States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.*

- `std::atomic_bool * mBackendsFoundAnswer`

*This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.*

- `Conditionals mFoundAnswer`

*Vector of Booleans: If any of them is true, we have to terminate a running check procedure.*

- `std::vector<Module *> mUsedBackends`

*The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).*

- `std::vector<Module *> mAllBackends`

*The backends of this module which have been used.*

- `std::vector<Lemma> mLemmas`

*Stores the lemmas being valid formulas this module or its backends made.*

- `ModuleInput::iterator mFirstSubformulaToPass`

*Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.*

- `carl::FastSet<FormulaT> mConstraintsToInform`

*Stores the constraints which the backends must be informed about.*

- `carl::FastSet<FormulaT> mInformedConstraints`

*Stores the position of the first constraint of which no backend has been informed about.*

- `ModuleInput::const_iterator mFirstUncheckedReceivedSubformula`

*Stores the position of the first (by this module) unchecked sub-formula of the received formula.*

- `unsigned mSmallerMusesCheckCounter`

*Counter used for the generation of the smt2 files to check for smaller muses.*

- `std::vector<std::size_t> mVariableCounters`

*Maps variables to the number of their occurrences.*

### 0.15.608.1 Member Typedef Documentation

**0.15.608.1.1 SettingsType** template<class Settings >  
 typedef `Settings smtrat::VSModule< Settings >::SettingsType`

### 0.15.608.2 Member Enumeration Documentation

**0.15.608.2.1 LemmaType** enum `smtrat::Module::LemmaType` : unsigned [strong], [inherited]

Enumerator

|           |  |
|-----------|--|
| NORMAL    |  |
| PERMANENT |  |

### 0.15.608.3 Constructor & Destructor Documentation

**0.15.608.3.1 VSModule()** template<class Settings >  
`smtrat::VSModule< Settings >::VSModule` (  
 const `ModuleInput` \* ,  
`Conditionals` & ,  
`Manager` \* const = `NULL` )

**0.15.608.3.2 ~VSModule()** template<class Settings >  
`smtrat::VSModule< Settings >::~VSModule` ( )

### 0.15.608.4 Member Function Documentation

**0.15.608.4.1 add()** bool `smtrat::Module::add` (  
`ModuleInput::const_iterator` `_subformula`) [inherited]

The module has to take the given sub-formula of the received formula into account.

Parameters

|                          |                                                    |
|--------------------------|----------------------------------------------------|
| <code>_subformula</code> | The sub-formula to take additionally into account. |
|--------------------------|----------------------------------------------------|

Returns

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

**0.15.608.4.2 addConstraintToInform()** void `smtrat::Module::addConstraintToInform` (  
`const FormulaT & _constraint`) [protected], [virtual], [inherited]

Adds a constraint to the collection of constraints of this module, which are informed to a freshly generated backend.

**Parameters**

|                          |                        |
|--------------------------|------------------------|
| <code>_constraint</code> | The constraint to add. |
|--------------------------|------------------------|

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.608.4.3 `addCore()`** `template<class Settings >`

```
bool smtrat::VSModule< Settings >::addCore (
 ModuleInput::const_iterator formula) [virtual]
```

The module has to take the given sub-formula of the received formula into account.

**Parameters**

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <code>formula</code> | The sub-formula to take additionally into account. |
|----------------------|----------------------------------------------------|

**Returns**

false, if it can be easily decided that this sub-formula causes a conflict with the already considered sub-formulas; true, otherwise.

Reimplemented from [smtrat::Module](#).

**0.15.608.4.4 `addInformationRelevantFormula()`** `void smtrat::Module::addInformationRelevantFormula`

```
(
 const FormulaT & formula) [protected], [inherited]
```

Adds a formula to the InformationRelevantFormula.

**Parameters**

|                      |                |
|----------------------|----------------|
| <code>formula</code> | Formula to add |
|----------------------|----------------|

**0.15.608.4.5 `addLemma()`** `void smtrat::Module::addLemma`

```
(
 const FormulaT & _lemma,
 const LemmaType & _lt = LemmaType::NORMAL,
 const FormulaT & _preferredFormula = FormulaT(carl::FormulaType::TRUE)) [inline],
[inherited]
```

Stores a lemma being a valid formula.

**Parameters**

|                                |                                                                       |
|--------------------------------|-----------------------------------------------------------------------|
| <code>_lemma</code>            | The eduction/lemma to store.                                          |
| <code>_lt</code>               | The type of the lemma.                                                |
| <code>_preferredFormula</code> | Hint for the next decision, which formula should be assigned to true. |

**0.15.608.4.6 `addOrigin()`** `void smtrat::Module::addOrigin`

```
(
 ModuleInput::iterator _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given set of formulas in the received formula to the origins of the given passed formula.

**Parameters**

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>_formula</code> | The passed formula to set the origins for.                |
| <code>_origin</code>  | A set of formulas in the received formula of this module. |

**0.15.608.4.7 addReceivedSubformulaToPassedFormula()** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addReceivedSubformulaToPassedFormula (
 ModuleInput::const_iterator _subformula) [inline], [protected], [inherited]
```

Copies the given sub-formula of the received formula to the passed formula.

Note, that there is always a link between sub-formulas of the passed formulas to sub-formulas of the received formulas, which are responsible for its occurrence.

**Parameters**

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| <code>_subformula</code> | The sub-formula of the received formula to copy. |
|--------------------------|--------------------------------------------------|

**0.15.608.4.8 addSubformulaToPassedFormula() [1/3]** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula with no origin.

Note that in the next call of this module's removeSubformula, all formulas in the passed formula without origins will be removed.

**Parameters**

|                       |                                           |
|-----------------------|-------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula. |
|-----------------------|-------------------------------------------|

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

**0.15.608.4.9 addSubformulaToPassedFormula() [2/3]** `std::pair<ModuleInput::iterator,bool>`

```
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const FormulaT & _origin) [inline], [protected], [inherited]
```

Adds the given formula to the passed formula.

**Parameters**

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                 |
| <code>_origin</code>  | The sub-formula of the received formula being responsible for the occurrence of the formula to add to the passed formula. |

**Returns**

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.608.4.10 addSubformulaToPassedFormula() [3/3] std::pair<ModuleInput::iterator,bool>
smrat::Module::addSubformulaToPassedFormula (
 const FormulaT & _formula,
 const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

Adds the given formula to the passed formula.

#### Parameters

|                       |                                                                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula to add to the passed formula.                                                                                               |
| <code>_origins</code> | The link of the formula to add to the passed formula to sub-formulas of the received formulas, which are responsible for its occurrence |

#### Returns

A pair to the position where the formula to add has been inserted (or its first sub-formula which has not yet been in the passed formula, in case the formula to add is a conjunction), and a Boolean stating whether anything has been added to the passed formula.

```
0.15.608.4.11 allModels() const std::vector<Model>& smrat::Module::allModels () const [inline],
[inherited]
```

#### Returns

All satisfying assignments, if existent.

```
0.15.608.4.12 anAnswerFound() bool smrat::Module::anAnswerFound () const [inline], [protected],
[inherited]
```

Checks for all antecedent modules and those which run in parallel with the same antecedent modules, whether one of them has determined a result.

#### Returns

True, if one of them has determined a result.

```
0.15.608.4.13 answerFound() const smrat::Conditionals& smrat::Module::answerFound () const
[inline], [inherited]
```

#### Returns

A vector of Booleans: If any of them is true, we have to terminate a running check procedure.

```
0.15.608.4.14 backendsModel() const Model & smrat::Module::backendsModel () const [protected],
[inherited]
```

Stores the model of a backend which determined satisfiability of the passed formula in the model of this module.

```
0.15.608.4.15 branchAt() [1/4] bool smtrat::Module::branchAt (
 carl::Variable::Arg _var,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

```
0.15.608.4.16 branchAt() [2/4] bool smtrat::Module::branchAt (
 carl::Variable::Arg _var,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [inline], [protected], [inherited]
```

```
0.15.608.4.17 branchAt() [3/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false,
 const std::vector< FormulaT > & _premise = std::vector<FormulaT>()) [inline],
[protected], [inherited]
```

```
0.15.608.4.18 branchAt() [4/4] bool smtrat::Module::branchAt (
 const Poly & _polynomial,
 bool _integral,
 const Rational & _value,
 std::vector< FormulaT > && _premise,
 bool _leftCaseWeak = true,
 bool _preferLeftCase = true,
 bool _useReceivedFormulaAsPremise = false) [protected], [inherited]
```

Adds a lemmas which provoke a branching for the given variable at the given value, if this module returns Unknown and there exists a preceding [SATModule](#).

Note that the given value is rounded down and up, if the given variable is integer-valued.

#### Parameters

|                                           |                                                                                                                                                                                                                |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_polynomial</code>                  | The variable to branch for.                                                                                                                                                                                    |
| <code>_integral</code>                    | A flag being true, if all variables in the polynomial to branch for are integral.                                                                                                                              |
| <code>_value</code>                       | The value to branch at.                                                                                                                                                                                        |
| <code>_premise</code>                     | The sub-formulas of the received formula from which the branch is followed. Note, that a premise is not necessary, as every branch is a valid formula. But a premise can prevent from branching unnecessarily. |
| <code>_leftCaseWeak</code>                | true, if a branching in the form of (or ( $\leq p b$ ) ( $> p b$ )) is desired. false, if a branching in the form of (or ( $< p b$ ) ( $\geq p b$ )) is desired.                                               |
| <code>_preferLeftCase</code>              | true, if the left case (polynomial less(or equal) 0 shall be chosen first. false, otherwise.                                                                                                                   |
| <code>_useReceivedFormulaAsPremise</code> | true, if the whole received formula should be used as premise                                                                                                                                                  |

```
0.15.608.4.19 check() Answer smtrat::Module::check (
 bool _final = false,
 bool _full = true,
 carl::Variable _objective = carl::Variable::NO_VARIABLE) [virtual], [inherited]
```

Checks the received formula for consistency.

Note, that this is an implementation of the satisfiability check of the conjunction of the so far received formulas, which does actually nothing but passing the problem to its backends. This implementation is only used internally and must be overwritten by any derived module.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the received formula is satisfiable; False, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

```
0.15.608.4.20 checkCore() template<class Settings >
Answer smtrat::VSModule< Settings >::checkCore () [virtual]
```

Checks the received formula for consistency.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT.                               |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

SAT, if the received formula is satisfiable; UNSAT, if the received formula is not satisfiable; Unknown, otherwise.

Reimplemented from [smtrat::Module](#).

```
0.15.608.4.21 checkInfSubsetForMinimality() void smtrat::Module::checkInfSubsetForMinimality (
 std::vector< FormulaSetT >::const_iterator _infssubset,
 const std::string & _filename = "smaller_muses",
 unsigned _maxSizeDifference = 1) const [inherited]
```

Checks the given infeasible subset for minimality by storing all subsets of it, which have a smaller size which differs from the size of the infeasible subset not more than the given threshold.

#### Parameters

|                           |                                                                                                        |
|---------------------------|--------------------------------------------------------------------------------------------------------|
| <i>_infssubset</i>        | The infeasible subset to check for minimality.                                                         |
| <i>_filename</i>          | The name of the file to store the infeasible subsets in order to be able to check their infeasibility. |
| <i>_maxSizeDifference</i> | The maximal difference between the sizes of the subsets compared to the size of the infeasible subset. |

**0.15.608.4.22 `checkModel()`** `unsigned smtrat::Module::checkModel ( ) const [protected], [inherited]`

**Returns**

false, if the current model of this module does not satisfy the current given formula; true, if it cannot be said whether the model satisfies the given formula.

**0.15.608.4.23 `cleanModel()`** `void smtrat::Module::cleanModel ( ) const [inline], [protected], [inherited]`

Substitutes variable occurrences by its model value in the model values of other variables.

**0.15.608.4.24 `clearLemmas()`** `void smtrat::Module::clearLemmas ( ) [inline], [inherited]`  
Deletes all yet found lemmas.

**0.15.608.4.25 `clearModel()`** `void smtrat::Module::clearModel ( ) const [inline], [protected], [inherited]`

Clears the assignment, if any was found.

**0.15.608.4.26 `clearModels()`** `void smtrat::Module::clearModels ( ) const [inline], [protected], [inherited]`

Clears all assignments, if any was found.

**0.15.608.4.27 `clearPassedFormula()`** `void smtrat::Module::clearPassedFormula ( ) [protected], [inherited]`

**0.15.608.4.28 `collectOrigins() [1/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulaSetT & _origins ) const [inherited]`

**0.15.608.4.29 `collectOrigins() [2/2]`** `void smtrat::Module::collectOrigins ( const FormulaT & _formula, FormulasT & _origins ) const [inherited]`

Collects the formulas in the given formula, which are part of the received formula.

If the given formula directly occurs in the received formula, it is inserted into the given set. Otherwise, the given formula must be of type AND and all its sub-formulas part of the received formula. Hence, they will be added to the given set.

**Parameters**

|                       |                                                                                                      |
|-----------------------|------------------------------------------------------------------------------------------------------|
| <code>_formula</code> | The formula from which to collect the formulas being sub-formulas of the received formula (origins). |
| <code>_origins</code> | The set in which to store the origins.                                                               |

**0.15.608.4.30 `collectTheoryPropagations()`** `void smtrat::Module::collectTheoryPropagations ( ) [inherited]`

**0.15.608.4.31 constraintsToInform()** const carl::FastSet<[FormulaT](#)>& smrat::Module::constraintsToInform( ) const [inline], [inherited]

**Returns**

The constraints which the backends must be informed about.

**0.15.608.4.32 currentlySatisfied()** virtual unsigned smrat::Module::currentlySatisfied( const [FormulaT](#) & ) const [inline], [virtual], [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise; 3, if we do not know anything (default)

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.608.4.33 currentlySatisfiedByBackend()** unsigned smrat::Module::currentlySatisfiedByBackend( const [FormulaT](#) & \_formula ) const [inherited]

**Returns**

0, if the given formula is conflicted by the current model; 1, if the given formula is satisfied by the current model; 2, otherwise 3, if we do not know anything (default)

**0.15.608.4.34 deinform()** void smrat::Module::deinform( const [FormulaT](#) & \_constraint ) [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

**0.15.608.4.35 deinformCore()** virtual void smrat::Module::deinformCore( const [FormulaT](#) & \_constraint ) [inline], [protected], [virtual], [inherited]

The inverse of informing about a constraint.

All data structures which were kept regarding this constraint are going to be removed. Note, that this makes only sense if it is not likely enough that a formula with this constraint must be solved again.

**Parameters**

|                          |                                                         |
|--------------------------|---------------------------------------------------------|
| <code>_constraint</code> | The constraint to remove from internal data structures. |
|--------------------------|---------------------------------------------------------|

Reimplemented in [smrat::LRAbstractModule< Settings >](#), [smrat::LRAbstractModule< LRASettingsICP >](#), and [smrat::LRAbstractModule< LRASettings >](#)

**0.15.608.4.36 determine\_smallest\_origin()** size\_t smrat::Module::determine\_smallest\_origin( const std::vector< [FormulaT](#) > & origins ) const [protected], [inherited]

## Parameters

|                       |                              |
|-----------------------|------------------------------|
| <code>_origins</code> | A vector of sets of origins. |
|-----------------------|------------------------------|

## Returns

The index of the smallest (regarding the size of the sets) element of origins

**0.15.608.4.37 `eraseSubformulaFromPassedFormula()`** `ModuleInput::iterator smtrat::Module::erase<SubformulaFromPassedFormula (`

```
 ModuleInput::iterator _subformula,
 bool _ignoreOrigins = false) [protected], [virtual], [inherited]
```

Removes everything related to the sub-formula to remove from the passed formula in the backends of this module. Afterwards the sub-formula is removed from the passed formula.

## Parameters

|                             |                                                                                                              |
|-----------------------------|--------------------------------------------------------------------------------------------------------------|
| <code>_subformula</code>    | The sub-formula to remove from the passed formula.                                                           |
| <code>_ignoreOrigins</code> | True, if the sub-formula shall be removed regardless of its origins (should only be applied with expertise). |

## Returns

Reimplemented in `smtrat::ICPModule< Settings >`, `smtrat::ICPModule< ICPSettings4 >`, and `smtrat::ICPModule< ICPSettings1 >`.

**0.15.608.4.38 `excludeNotReceivedVariablesFromModel()`** `void smtrat::Module::excludeNotReceived<VariablesFromModel ( ) const [inherited]`

Excludes all variables from the current model, which do not occur in the received formula.

**0.15.608.4.39 `findBestOrigin()`** `std::vector< FormulaT >::const_iterator smtrat::Module::find<BestOrigin (`

```
 const std::vector< FormulaT > & _origins) const [protected], [inherited]
```

## Parameters

|                       |  |
|-----------------------|--|
| <code>_origins</code> |  |
|-----------------------|--|

## Returns

**0.15.608.4.40 `firstSubformulaToPass()`** `ModuleInput::const_iterator smtrat::Module::firstSubformula<ToPass ( ) const [inline], [inherited]`

## Returns

The position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.608.4.41 firstUncheckedReceivedSubformula()** `ModuleInput::const_iterator smtrat::Module::firstUncheckedReceivedSubformula ( ) const [inline], [inherited]`

**Returns**

The position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.608.4.42 freeSplittingVariable()** `static void smtrat::Module::freeSplittingVariable ( const FormulaT & _splittingVariable ) [inline], [static], [inherited]`

**0.15.608.4.43 generateTrivialInfeasibleSubset()** `void smtrat::Module::generateTrivialInfeasibleSubset ( ) [inline], [protected], [inherited]`

Stores the trivial infeasible subset being the set of received formulas.

**0.15.608.4.44 getBackendsAllModels()** `void smtrat::Module::getBackendsAllModels ( ) const [protected], [inherited]`

Stores all models of a backend in the list of all models of this module.

**0.15.608.4.45 getBackendsModel()** `void smtrat::Module::getBackendsModel ( ) const [protected], [inherited]`

**0.15.608.4.46 getInfeasibleSubsets() [1/2]** `void smtrat::Module::getInfeasibleSubsets ( ) [protected], [inherited]`

Copies the infeasible subsets of the passed formula.

**0.15.608.4.47 getInfeasibleSubsets() [2/2]** `std::vector< FormulaSetT > smtrat::Module::getInfeasibleSubsets (`

`const Module & _backend ) const [protected], [inherited]`

Get the infeasible subsets the given backend provides.

Note, that an infeasible subset in a backend contains sub formulas of the passed formula and an infeasible subset of this module contains sub formulas of the received formula. In this method the LATTER is returned.

**Parameters**

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| <code>_backend</code> | The backend from which to obtain the infeasible subsets. |
|-----------------------|----------------------------------------------------------|

**Returns**

The infeasible subsets the given backend provides.

**0.15.608.4.48 getInformationRelevantFormulas()** `const std::set< FormulaT > & smtrat::Module::getInformationRelevantFormulas ( ) [protected], [inherited]`

Gets all InformationRelevantFormulas.

**Returns**

Set of all formulas

**0.15.608.4.49 getModelEqualities()** `std::list< std::vector< carl::Variable > > smtrat::Module::getModelEqualities ( ) const [virtual], [inherited]`

Partition the variables from the current model into equivalence classes according to their assigned value.

The result is a set of equivalence classes of variables where all variables within one class are assigned the same value. Note that the number of classes may not be minimal, i.e. two classes may actually be equivalent.

#### Returns

Equivalence classes.

**0.15.608.4.50 getOrigins() [1/3]** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulaSetT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.608.4.51 getOrigins() [2/3]** `void smtrat::Module::getOrigins (`

```
const FormulaT & _formula,
FormulasT & _origins) const [inline], [protected], [inherited]
```

#### Parameters

|                       |  |
|-----------------------|--|
| <code>_formula</code> |  |
| <code>_origins</code> |  |

**0.15.608.4.52 getOrigins() [3/3]** `const FormulaT& smtrat::Module::getOrigins (`

```
ModuleInput::const_iterator _formula) const [inline], [protected], [inherited]
```

Gets the origins of the passed formula at the given position.

#### Parameters

|                       |                                                   |
|-----------------------|---------------------------------------------------|
| <code>_formula</code> | The position of a formula in the passed formulas. |
|-----------------------|---------------------------------------------------|

#### Returns

The origins of the passed formula at the given position.

**0.15.608.4.53 getReceivedFormulaSimplified()** `std::pair< bool, FormulaT > smtrat::Module::getReceivedFormulaSimplified ( ) [virtual], [inherited]`

#### Returns

A pair of a Boolean and a formula, where the Boolean is true, if the received formula could be simplified to an equisatisfiable formula. The formula is equisatisfiable to this module's received formula, if the Boolean is true.

Reimplemented in [smtrat::PModule](#).

**0.15.608.4.54 hasLemmas()** `bool smrat::Module::hasLemmas () [inline], [inherited]`  
Checks whether this module or any of its backends provides any lemmas.

**0.15.608.4.55 hasValidInfeasibleSubset()** `bool smrat::Module::hasValidInfeasibleSubset () const [inherited]`

**Returns**

true, if the module has at least one valid infeasible subset, that is all its elements are sub-formulas of the received formula (pointer must be equal).

**0.15.608.4.56 id()** `std::size_t smrat::Module::id () const [inline], [inherited]`

**Returns**

A unique ID to identify this module instance.

**0.15.608.4.57 infeasibleSubsets()** `const std::vector<FormulaSetT>& smrat::Module::infeasibleSubsets () const [inline], [inherited]`

**Returns**

The infeasible subsets of the set of received formulas (empty, if this module has not detected unsatisfiability of the conjunction of received formulas).

**0.15.608.4.58 inform()** `bool smrat::Module::inform (const FormulaT & _constraint) [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**Returns**

false, if it can be easily decided whether the given constraint is inconsistent; true, otherwise.

**0.15.608.4.59 informBackends()** `void smrat::Module::informBackends (const FormulaT & _constraint) [inline], [protected], [inherited]`

Informs all backends of this module about the given constraint.

**Parameters**

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

**0.15.608.4.60 informCore()** `virtual bool smrat::Module::informCore (const FormulaT & _constraint) [inline], [protected], [virtual], [inherited]`

Informs the module about the given constraint.

It should be tried to inform this module about any constraint it could receive eventually before assertSubformula is called (preferably for the first time, but at least before adding a formula containing that constraint).

#### Parameters

|                          |                                 |
|--------------------------|---------------------------------|
| <code>_constraint</code> | The constraint to inform about. |
|--------------------------|---------------------------------|

#### Returns

`false`, if it can be easily decided whether the given constraint is inconsistent; `true`, otherwise.

Reimplemented in [smrat::UnionFindModule< Settings >](#), [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), [smrat::BVMModule< Settings >](#), [smrat::ICPModule< Settings >](#), [smrat::ICPModule< ICPSettings4 >](#), and [smrat::ICPModule< ICPSettings1 >](#).

**0.15.608.4.61 informedConstraints()** `const carl::FastSet<FormulaT>& smrat::Module::informedConstraints( ) const [inline], [inherited]`

#### Returns

The position of the first constraint of which no backend has been informed about.

**0.15.608.4.62 init()** `void smrat::Module::init( ) [virtual], [inherited]`

Informs all backends about the so far encountered constraints, which have not yet been communicated.

This method must not be called twice and only before the first runBackends call.

Reimplemented in [smrat::PBPPModule< Settings >](#), [smrat::PBGaussModule< Settings >](#), [smrat::NRAILModule< Settings >](#), [smrat::NewCoveringModule< Settings >](#), [smrat::NewCADModule< Settings >](#), [smrat::LRAModule< Settings >](#), [smrat::LRAModule< LRASettingsICP >](#), [smrat::LRAModule< LRASettings1 >](#), [smrat::IntBlastModule< Settings >](#), [smrat::FPPModule< Settings >](#), [smrat::EQModule< Settings >](#), [smrat::CurryModule< Settings >](#), and [smrat::BVMModule< Settings >](#).

**0.15.608.4.63 is\_minimizing()** `bool smrat::Module::is_minimizing( ) const [inline], [inherited]`

**0.15.608.4.64 isLemmaLevel()** `bool smrat::Module::isLemmaLevel(`

`LemmaLevel level) [protected], [inherited]`

Checks if current lemma level is greater or equal to given level.

#### Parameters

|                    |                 |
|--------------------|-----------------|
| <code>level</code> | Level to check. |
|--------------------|-----------------|

**0.15.608.4.65 isPreprocessor()** `bool smrat::Module::isPreprocessor( ) const [inline], [inherited]`

#### Returns

`true`, if this module is a preprocessor that is a module, which simplifies its received formula to an equisatisfiable formula being passed to its backends. The simplified formula can be obtained with [getReceivedFormulaSimplified\(\)](#).

**0.15.608.4.66 lemmas()** const std::vector<[Lemma](#)>& smtrat::Module::lemmas ( ) const [inline], [inherited]

**Returns**

A constant reference to the lemmas being valid formulas this module or its backends made.

**0.15.608.4.67 merge()** std::vector< [FormulaT](#) > smtrat::Module::merge ( const std::vector< [FormulaT](#) > & [\\_vecSetA](#), const std::vector< [FormulaT](#) > & [\\_vecSetB](#) ) const [protected], [inherited]

Merges the two vectors of sets into the first one.

({a,b},{a,c}) and ({b,d},{b}) -> ({a,b,d},{a,b},{a,b,c,d},{a,b,c})

**Parameters**

|                          |                                  |
|--------------------------|----------------------------------|
| <a href="#">_vecSetA</a> | A vector of sets of constraints. |
| <a href="#">_vecSetB</a> | A vector of sets of constraints. |

**Returns**

The vector being the two given vectors merged.

**0.15.608.4.68 model()** const [Model](#)& smtrat::Module::model ( ) const [inline], [inherited]

**Returns**

The assignment of the current satisfiable result, if existent.

**0.15.608.4.69 modelsDisjoint()** bool smtrat::Module::modelsDisjoint ( const [Model](#) & [\\_modelA](#), const [Model](#) & [\\_modelB](#) ) [static], [protected], [inherited]

Checks whether there is no variable assigned by both models.

**Parameters**

|                         |                                |
|-------------------------|--------------------------------|
| <a href="#">_modelA</a> | The first model to check for.  |
| <a href="#">_modelB</a> | The second model to check for. |

**Returns**

true, if there is no variable assigned by both models; false, otherwise.

**0.15.608.4.70 moduleName()** template<class [Settings](#) > std::string smtrat::VSModule< [Settings](#) >::moduleName ( ) const [inline], [virtual]

**Returns**

The name of the given module type as name.

Reimplemented from [smtrat::Module](#).

**0.15.608.4.71 `objective()`** `carl::Variable smrat::Module::objective ( ) const [inline], [inherited]`

**0.15.608.4.72 `originInReceivedFormula()`** `bool smrat::Module::originInReceivedFormula ( const FormulaT & _origin ) const [protected], [inherited]`

**0.15.608.4.73 `passedFormulaBegin()`** `ModuleInput::iterator smrat::Module::passedFormulaBegin ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula. TODO: disable this method

**0.15.608.4.74 `passedFormulaEnd()`** `ModuleInput::iterator smrat::Module::passedFormulaEnd ( ) [inline], [protected], [inherited]`

**Returns**

An iterator to the end of the passed formula.

**0.15.608.4.75 `pPassedFormula()`** `const ModuleInput* smrat::Module::pPassedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to the backends of this module.

**0.15.608.4.76 `pReceivedFormula()`** `const ModuleInput* smrat::Module::pReceivedFormula ( ) const [inline], [inherited]`

**Returns**

A pointer to the formula passed to this module.

**0.15.608.4.77 `print()`** `void smrat::Module::print ( const std::string & _initiation = "***" ) const [inherited]`

Prints everything relevant of the solver.

**Parameters**

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.608.4.78 `printAll()`** `template<class Settings > void smrat::VSModule< Settings >::printAll ( const std::string & _init = "", std::ostream & _out = std::cout ) const`

Prints the history to the output stream.

**Parameters**

|              |                                                        |
|--------------|--------------------------------------------------------|
| <i>_init</i> | The beginning of each row.                             |
| <i>_out</i>  | The output stream where the history should be printed. |

**0.15.608.4.79 printAllModels()** void smtrat::Module::printAllModels ( std::ostream & *\_out* = std::cout ) [inherited]

Prints all assignments of this module satisfying its received formula if it satisfiable and this module could find an assignment.

**Parameters**

|             |                                        |
|-------------|----------------------------------------|
| <i>_out</i> | The stream to print the assignment on. |
|-------------|----------------------------------------|

**0.15.608.4.80 printAnswer()** template<class Settings >  
void smtrat::VSModule< Settings >::printAnswer ( const std::string & *\_init* = "", std::ostream & *\_out* = std::cout ) const

Prints the answer if existent.

**Parameters**

|              |                                                       |
|--------------|-------------------------------------------------------|
| <i>_init</i> | The beginning of each row.                            |
| <i>_out</i>  | The output stream where the answer should be printed. |

**0.15.608.4.81 printFormulaConditionMap()** template<class Settings >  
void smtrat::VSModule< Settings >::printFormulaConditionMap ( const std::string & *\_init* = "", std::ostream & *\_out* = std::cout ) const

Prints the history to the output stream.

**Parameters**

|              |                                                        |
|--------------|--------------------------------------------------------|
| <i>_init</i> | The beginning of each row.                             |
| <i>_out</i>  | The output stream where the history should be printed. |

**0.15.608.4.82 printInfeasibleSubsets()** void smtrat::Module::printInfeasibleSubsets ( const std::string & *\_initiation* = "\*\*\*" ) const [inherited]

Prints the infeasible subsets.

**Parameters**

|                    |                      |
|--------------------|----------------------|
| <i>_initiation</i> | The line initiation. |
|--------------------|----------------------|

**0.15.608.4.83 printModel()** void smtrat::Module::printModel (

```
std::ostream & _out = std::cout) const [inherited]
```

Prints the assignment of this module satisfying its received formula if it satisfiable and this module could find an assignment.

#### Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <code>_out</code> | The stream to print the assignment on. |
|-------------------|----------------------------------------|

**0.15.608.4.84 `printPassedFormula()`** void smtrat::Module::printPassedFormula ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]

Prints the vector of passed formula.

#### Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.608.4.85 `printRanking()`** template<class Settings > void smtrat::VSModule< Settings >::printRanking ( const std::string & `_init` = "", std::ostream & `_out` = std::cout ) const

Prints the history to the output stream.

#### Parameters

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| <code>_init</code> | The beginning of each row.                             |
| <code>_out</code>  | The output stream where the history should be printed. |

**0.15.608.4.86 `printReceivedFormula()`** void smtrat::Module::printReceivedFormula ( const std::string & `_initiation` = "\*\*\*" ) const [inherited]

Prints the vector of the received formula.

#### Parameters

|                          |                      |
|--------------------------|----------------------|
| <code>_initiation</code> | The line initiation. |
|--------------------------|----------------------|

**0.15.608.4.87 `probablyLooping()`** bool smtrat::Module::probablyLooping ( const typename Poly::PolyType & `_branchingPolynomial`, const Rational & `_branchingValue` ) const [protected], [inherited]

Checks whether given the current branching value and branching variable/polynomial it is (highly) probable that this branching is part of an infinite loop of branchings.

#### Parameters

|                                   |                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------|
| <code>_branchingPolynomial</code> | The polynomial to branch at (in most branching strategies this is just a variable). |
| <code>_branchingValue</code>      | The value to branch at.                                                             |

**Returns**

true, if this branching is probably part of an infinite loop of branchings; false, otherwise.

**0.15.608.4.88 receivedFormulaChecked()** void smtrat::Module::receivedFormulaChecked () [inline], [inherited]

Notifies that the received formulas has been checked.

**0.15.608.4.89 receivedFormulasAsInfeasibleSubset()** void smtrat::Module::receivedFormulasAsInfeasibleSubset (

    ModuleInput::const\_iterator \_subformula ) [inline], [protected], [inherited]

Stores an infeasible subset consisting only of the given received formula.

**0.15.608.4.90 receivedVariable()** bool smtrat::Module::receivedVariable (

    carl::Variable::Arg \_var ) const [inline], [inherited]

**0.15.608.4.91 remove()** void smtrat::Module::remove (

    ModuleInput::const\_iterator \_subformula ) [virtual], [inherited]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|             |                                                    |
|-------------|----------------------------------------------------|
| _subformula | The sub formula of the received formula to remove. |
|-------------|----------------------------------------------------|

Reimplemented in [smtrat::PModule](#).

**0.15.608.4.92 removeCore()** template<class Settings >

void smtrat::VSModule< Settings >::removeCore (

    ModuleInput::const\_iterator formula ) [virtual]

Removes everything related to the given sub-formula of the received formula.

However, it is desired not to lose track of search spaces where no satisfying assignment can be found for the remaining sub-formulas.

**Parameters**

|         |                                                    |
|---------|----------------------------------------------------|
| formula | The sub formula of the received formula to remove. |
|---------|----------------------------------------------------|

Reimplemented from [smtrat::Module](#).

**0.15.608.4.93 removeOrigin()** std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigin (

    ModuleInput::iterator formula,

    const FormulaT & \_origin ) [inline], [protected], [inherited]

**0.15.608.4.94 removeOrigins()** std::pair<ModuleInput::iterator,bool> smtrat::Module::removeOrigins (

    ModuleInput::iterator formula,

```
const std::shared_ptr< std::vector< FormulaT >> & _origins) [inline], [protected],
[inherited]
```

**0.15.608.4.95 rPassedFormula()** const ModuleInput& smtrat::Module::rPassedFormula ( ) const  
[inline], [inherited]

#### Returns

A reference to the formula passed to the backends of this module.

**0.15.608.4.96 rReceivedFormula()** const ModuleInput& smtrat::Module::rReceivedFormula ( ) const  
[inline], [inherited]

#### Returns

A reference to the formula passed to this module.

**0.15.608.4.97 runBackends() [1/2]** virtual Answer smtrat::Module::runBackends ( ) [inline],  
[protected], [virtual], [inherited]  
Reimplemented in [smtrat::PModule](#).

**0.15.608.4.98 runBackends() [2/2]** Answer smtrat::Module::runBackends (

```
 bool _final,
 bool _full,
 carl::Variable _objective) [protected], [virtual], [inherited]
```

Runs the backend solvers on the passed formula.

#### Parameters

|                   |                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>_final</i>     | true, if this satisfiability check will be the last one (for a global sat-check), if its result is SAT                                |
| <i>_full</i>      | false, if this module should avoid too expensive procedures and rather return unknown instead.                                        |
| <i>_objective</i> | if not set to NO_VARIABLE, the module should find an assignment minimizing this objective variable; otherwise any assignment is good. |

#### Returns

True, if the passed formula is consistent; False, if the passed formula is inconsistent; Unknown, otherwise.

Reimplemented in [smtrat::PModule](#).

**0.15.608.4.99 setId()** void smtrat::Module::setId ( std::size\_t *\_id* ) [inline], [inherited]

Sets this modules unique ID to identify itself.

#### Parameters

|                                                            |                                    |
|------------------------------------------------------------|------------------------------------|
| $\leftarrow$<br>$\overleftarrow{\rightarrow}$<br><i>id</i> | The id to set this module's id to. |
|------------------------------------------------------------|------------------------------------|

**0.15.608.4.100 setThreadPriority()** `void smtrat::Module::setThreadPriority (``thread_priority _threadPriority ) [inline], [inherited]`

Sets the priority of this module to get a thread for running its check procedure.

**Parameters**

|                              |                                                       |
|------------------------------|-------------------------------------------------------|
| <code>_threadPriority</code> | The priority to set this module's thread priority to. |
|------------------------------|-------------------------------------------------------|

**0.15.608.4.101 solverState()** `Answer smtrat::Module::solverState ( ) const [inline], [inherited]`**Returns**

True, if this module is in a state, such that it has found a solution for its received formula; False, if this module is in a state, such that it has determined that there is no solution for its received formula; Unknown, otherwise.

**0.15.608.4.102 splitUnequalConstraint()** `void smtrat::Module::splitUnequalConstraint (``const FormulaT & _unequalConstraint ) [protected], [inherited]`

Adds the following lemmas for the given constraint  $p \neq 0$ .

$(p \neq 0 \leftrightarrow (p < 0 \text{ or } p > 0))$

and  $\neg(p < 0 \text{ and } p > 0)$

**Parameters**

|                                 |                                                  |
|---------------------------------|--------------------------------------------------|
| <code>_unequalConstraint</code> | A constraint having the relation symbol $\neq$ . |
|---------------------------------|--------------------------------------------------|

**0.15.608.4.103 threadPriority()** `thread_priority smtrat::Module::threadPriority ( ) const [inline], [inherited]`**Returns**

The priority of this module to get a thread for running its check procedure.

**0.15.608.4.104 updateAllModels()** `void smtrat::Module::updateAllModels ( ) [virtual], [inherited]`

Updates all satisfying models, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented in [smtrat::SATModule< Settings >](#).

**0.15.608.4.105 updateLemmas()** `void smtrat::Module::updateLemmas ( ) [inherited]`

Stores all lemmas of any backend of this module in its own lemma vector.

**0.15.608.4.106 updateModel()** `template<class Settings >``void smtrat::VSMModule< Settings >::updateModel ( ) const [virtual]`

Updates the model, if the solver has detected the consistency of the received formula, beforehand.

Reimplemented from [smtrat::Module](#).

**0.15.608.4.107 usedBackends()** `const std::vector<Module*>& smtrat::Module::usedBackends ( )``const [inline], [inherited]`

**Returns**

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.608.5 Field Documentation**

**0.15.608.5.1 mAllBackends** std::vector<[Module](#)\*> smtrat::Module::mAllBackends [protected], [inherited]

The backends of this module which have been used.

**0.15.608.5.2 mAllModels** std::vector<[Model](#)> smtrat::Module::mAllModels [mutable], [protected], [inherited]

Stores all satisfying assignments.

**0.15.608.5.3 mBackendsFoundAnswer** std::atomic\_bool\* smtrat::Module::mBackendsFoundAnswer [protected], [inherited]

This flag is passed to any backend and if it determines an answer to a prior check call, this flag is fired.

**0.15.608.5.4 mConstraintsToInform** carl::FastSet<[FormulaT](#)> smtrat::Module::mConstraintsToInform [protected], [inherited]

Stores the constraints which the backends must be informed about.

**0.15.608.5.5 mFinalCheck** bool smtrat::Module::mFinalCheck [protected], [inherited]  
true, if the check procedure should perform a final check which especially means not to postpone splitting decisions

**0.15.608.5.6 mFirstPosInLastBranches** std::size\_t smtrat::Module::mFirstPosInLastBranches = 0 [static], [inherited]

The beginning of the cyclic buffer storing the last branches.

**0.15.608.5.7 mFirstSubformulaToPass** [ModuleInput](#)::iterator smtrat::Module::mFirstSubformulaToPass [protected], [inherited]

Stores the position of the first sub-formula in the passed formula, which has not yet been considered for a consistency check of the backends.

**0.15.608.5.8 mFirstUncheckedReceivedSubformula** [ModuleInput](#)::const\_iterator smtrat::Module::mFirstUncheckedReceivedSubformula [protected], [inherited]

Stores the position of the first (by this module) unchecked sub-formula of the received formula.

**0.15.608.5.9 mFoundAnswer** [Conditionals](#) smtrat::Module::mFoundAnswer [protected], [inherited]  
Vector of Booleans: If any of them is true, we have to terminate a running check procedure.

**0.15.608.5.10 mFullCheck** bool smtrat::Module::mFullCheck [protected], [inherited]  
false, if this module should avoid too expensive procedures and rather return unknown instead.

**0.15.608.5.11 mInfeasibleSubsets** std::vector<[FormulaSetT](#)> smtrat::Module::mInfeasibleSubsets  
[protected], [inherited]  
Stores the infeasible subsets.

**0.15.608.5.12 mInformedConstraints** carl::FastSet<[FormulaT](#)> smtrat::Module::mInformedConstraints  
[protected], [inherited]  
Stores the position of the first constraint of which no backend has been informed about.

**0.15.608.5.13 mLastBranches** std::vector< [Branching](#) > smtrat::Module::mLastBranches = std::vector<[Branching](#)>(mNumOfBranchVarsToStore, Branching(typename Poly::PolyType(), 0)) [static], [inherited]  
Stores the last branches in a cycle buffer.

**0.15.608.5.14 mLemmas** std::vector<[Lemma](#)> smtrat::Module::mLemmas [protected], [inherited]  
Stores the lemmas being valid formulas this module or its backends made.

**0.15.608.5.15 mModel** [Model](#) smtrat::Module::mModel [mutable], [protected], [inherited]  
Stores the assignment of the current satisfiable result, if existent.

**0.15.608.5.16 mModelComputed** bool smtrat::Module::mModelComputed [mutable], [protected], [inherited]  
True, if the model has already been computed.

**0.15.608.5.17 mNumOfBranchVarsToStore** std::size\_t smtrat::Module::mNumOfBranchVarsToStore = 5 [static], [inherited]  
The number of different variables to consider for a probable infinite loop of branchings.

**0.15.608.5.18 mObjectiveVariable** carl::Variable smtrat::Module::mObjectiveVariable [protected], [inherited]  
Objective variable to be minimized. If set to NO\_VARIABLE, a normal sat check should be performed.

**0.15.608.5.19 mOldSplittingVariables** std::vector< [FormulaT](#) > smtrat::Module::mOldSplittingVariables [static], [inherited]  
Reusable splitting variables.

**0.15.608.5.20 mpManager** [Manager](#)\* const smtrat::Module::mpManager [protected], [inherited]  
A reference to the manager.

**0.15.608.5.21 mSmallerMusesCheckCounter** unsigned smtrat::Module::mSmallerMusesCheckCounter [mutable], [protected], [inherited]  
Counter used for the generation of the smt2 files to check for smaller muses.

**0.15.608.5.22 mSolverState** std::atomic<[Answer](#)> smtrat::Module::mSolverState [protected],  
 [inherited]

States whether the received formula is known to be satisfiable or unsatisfiable otherwise it is set to unknown.

**0.15.608.5.23 mTheoryPropagations** std::vector<[TheoryPropagation](#)> smtrat::Module::mTheory←  
 Propagations [protected], [inherited]

**0.15.608.5.24 mUsedBackends** std::vector<[Module\\*](#)> smtrat::Module::mUsedBackends [protected],  
 [inherited]

The backends of this module which are currently used (conditions to use this module are fulfilled for the passed formula).

**0.15.608.5.25 mVariableCounters** std::vector<std::size\_t> smtrat::Module::mVariableCounters  
 [protected], [inherited]

Maps variables to the number of their occurrences.

## 0.15.609 Minisat::Watcher Struct Reference

[[Minisat](#) related code]

```
#include <SolverTypes.h>
```

### Public Member Functions

- **Watcher** ([Minisat::CRef](#) cr, [Minisat::Lit](#) p)  
*[Minisat related code]*
- bool **operator==** (const [Watcher](#) &w) const  
*[Minisat related code]*
- bool **operator!=** (const [Watcher](#) &w) const  
*[Minisat related code]*

### Data Fields

- [Minisat::CRef](#) cref  
*[Minisat related code]*
- [Minisat::Lit](#) blocker  
*[Minisat related code]*

### Friends

- std::ostream & **operator<<** (std::ostream &os, const [Watcher](#) &w)

## 0.15.609.1 Detailed Description

[[Minisat](#) related code]

## 0.15.609.2 Constructor & Destructor Documentation

**0.15.609.2.1 Watcher()** [Minisat::Watcher](#)::Watcher (  
[Minisat::CRef](#) cr,  
[Minisat::Lit](#) p ) [inline]

[[Minisat](#) related code]

### 0.15.609.3 Member Function Documentation

**0.15.609.3.1 operator"!=()** bool Minisat::Watcher::operator!= ( const `Watcher` & w ) const [inline]  
[Minisat related code]

**0.15.609.3.2 operator==()** bool Minisat::Watcher::operator== ( const `Watcher` & w ) const [inline]  
[Minisat related code]

### 0.15.609.4 Friends And Related Function Documentation

**0.15.609.4.1 operator<<** std::ostream& operator<< ( std::ostream & os, const `Watcher` & w ) [friend]

### 0.15.609.5 Field Documentation

**0.15.609.5.1 blocker** `Minisat::Lit` Minisat::Watcher::blocker  
[Minisat related code]

**0.15.609.5.2 cref** `Minisat::CRef` Minisat::Watcher::cref  
[Minisat related code]

## 0.15.610 smtrat::ICPModule< Settings >::weights Struct Reference

#include <ICPModule.h>

### Data Fields

- std::list<`linearVariable` \* > origins
- double weight

### 0.15.610.1 Field Documentation

**0.15.610.1.1 origins** template<class Settings > std::list<`linearVariable`\*> smtrat::ICPModule< Settings >::weights::origins

**0.15.610.1.2 weight** template<class Settings > double smtrat::ICPModule< Settings >::weights::weight

## 0.15.611 benchmax::XMLWriter Class Reference

Writes results to a xml file.

#include <XMLWriter.h>

## Public Member Functions

- `XMLWriter` (const std::string &filename)  
*Open file for writing.*
- template<typename Results>  
`void write` (const `Jobs` &jobs, const `Results` &results)  
*Write results to file.*

### 0.15.611.1 Detailed Description

Writes results to a xml file.

### 0.15.611.2 Constructor & Destructor Documentation

**0.15.611.2.1 XMLWriter()** `benchmax::XMLWriter::XMLWriter` (  
`const std::string & filename`) [inline]

Open file for writing.

### 0.15.611.3 Member Function Documentation

**0.15.611.3.1 write()** template<typename Results>  
`void benchmax::XMLWriter::write` (  
`const Jobs & jobs,`  
`const Results & results`) [inline]

Write results to file.

## 0.15.612 `benchmax::Z3` Class Reference

Tool wrapper for `Z3` for SMT-LIB.

```
#include <Z3.h>
```

## Public Member Functions

- `Z3` (const fs::path &`binary`, const std::string &`arguments`)  
*New Z3 tool.*
- virtual bool `canHandle` (const fs::path &`path`) const override  
*Only handle .smt2 files.*
- virtual void `additionalResults` (const fs::path &, `BenchmarkResult` &`result`) const override  
*Parse answer string from stdout.*
- std::string `name` () const  
*Common name of this tool.*
- fs::path `binary` () const  
*Full path to the binary.*
- const std::map<std::string, std::string> & `attributes` () const  
*A set of attributes, for example compilation options.*
- std::vector<std::string> `resolveDependencies` () const  
*Get dependencies of binary required to run it (via ldd)*
- std::size\_t `attributeHash` () const  
*Hash of the attributes.*
- virtual std::string `getCommandline` (const std::string &`file`) const  
*Compose commandline for this tool and the given input file.*

- virtual std::string [getCommandline](#) (const std::string &file, const std::string &localBinary) const  
*Compose commandline for this tool with another binary name and the given input file.*
- virtual std::optional< std::string > [parseCommandline](#) (const std::string &cmdline) const  
*Compose commandline for this tool and the given input file.*

## Protected Attributes

- std::string [mName](#)  
*(Non-unique) identifier for the tool.*
- fs::path [mBinary](#)  
*Path to the binary.*
- std::string [mArguments](#)  
*Command line arguments that should be passed to the binary.*
- std::map< std::string, std::string > [mAttributes](#)  
*Attributes of the tool obtained by introspection of the binary.*

### 0.15.612.1 Detailed Description

Tool wrapper for [Z3](#) for SMT-LIB.

### 0.15.612.2 Constructor & Destructor Documentation

**0.15.612.2.1 [Z3\(\)](#)** benchmax::Z3::Z3 (

```
 const fs::path & binary,
 const std::string & arguments) [inline]
```

New [Z3](#) tool.

### 0.15.612.3 Member Function Documentation

**0.15.612.3.1 [additionalResults\(\)](#)** virtual void benchmax::Z3::additionalResults (

```
 const fs::path & ,
 BenchmarkResult & result) const [inline], [override], [virtual]
```

Parse answer string from stdout.

Reimplemented from [benchmax::Tool](#).

**0.15.612.3.2 [attributeHash\(\)](#)** std::size\_t benchmax::Tool::attributeHash () const [inline], [inherited]

Hash of the attributes.

**0.15.612.3.3 [attributes\(\)](#)** const std::map<std::string, std::string>& benchmax::Tool::attributes (

```
) const [inline], [inherited]
```

A set of attributes, for example compilation options.

**0.15.612.3.4 [binary\(\)](#)** fs::path benchmax::Tool::binary () const [inline], [inherited]

Full path to the binary.

**0.15.612.3.5 canHandle()** virtual bool benchmax::Z3::canHandle ( const fs::path & path ) const [inline], [override], [virtual]  
Only handle .smt2 files.  
Reimplemented from [benchmax::Tool](#).

**0.15.612.3.6 getCommandLine() [1/2]** virtual std::string benchmax::Tool::getCommandLine ( const std::string & file ) const [inline], [virtual], [inherited]  
Compose commandline for this tool and the given input file.

**0.15.612.3.7 getCommandLine() [2/2]** virtual std::string benchmax::Tool::getCommandLine ( const std::string & file, const std::string & localBinary ) const [inline], [virtual], [inherited]  
Compose commandline for this tool with another binary name and the given input file.

**0.15.612.3.8 name()** std::string benchmax::Tool::name () const [inline], [inherited]  
Common name of this tool.

**0.15.612.3.9 parseCommandLine()** virtual std::optional<std::string> benchmax::Tool::parseCommandline ( const std::string & cmdline ) const [inline], [virtual], [inherited]  
Compose commandline for this tool and the given input file.

**0.15.612.3.10 resolveDependencies()** std::vector<std::string> benchmax::Tool::resolveDependencies () const [inline], [inherited]  
Get dependencies of binary required to run it (via ldd)

## 0.15.612.4 Field Documentation

**0.15.612.4.1 mArguments** std::string benchmax::Tool::mArguments [protected], [inherited]  
Command line arguments that should be passed to the binary.

**0.15.612.4.2 mAttributes** std::map<std::string, std::string> benchmax::Tool::mAttributes [protected], [inherited]  
Attributes of the tool obtained by introspection of the binary.

**0.15.612.4.3 mBinary** fs::path benchmax::Tool::mBinary [protected], [inherited]  
Path to the binary.

**0.15.612.4.4 mName** std::string benchmax::Tool::mName [protected], [inherited]  
(Non-unique) identifier for the tool.

